Applicative theories for logarithmic complexity classes

Sebastian Eberhard

Institut für Informatik und angewandte Mathematik, Universität Bern, Neubrückstrasse 10, CH-3012 Bern, Switzerland.

Abstract

We present applicative theories of words corresponding to weak, and especially logarithmic, complexity classes. The theories for the logarithmic hierarchy and alternating logarithmic time formalise function algebras with concatenation recursion as main principle. We present two theories for logarithmic space where the first formalises a new two-sorted algebra which is very similar to Cook and Bellantoni's famous two-sorted algebra B for polynomial time [4]. The second theory describes logarithmic space by formalising concatenation - and sharply bounded recursion. All theories contain the predicates W representing words, and V representing temporary inaccessible words. They are inspired by Cantini's theories [6] formalising B.

1. Introduction

There are many examples of logical theories corresponding to complexity classes given by function algebras. Research on this subject was started by Buss in [5] where theories of bounded arithmetic are introduced for the subclasses of the polynomial hierarchy. Further theories of bounded arithmetic have been introduced by Clote and Takeuti in [8] amongst others for the logarithmic hierarchy, alternating logarithmic time, logarithmic space, and various circuit complexity classes. Weak arithmetic *second-order* theories corresponding to various complexity classes, analysed by various researchers, are collected in [9]. Weak second order theories have also been proposed by Buss [5], and Krajíček [17].

Let us switch now to applicative theories, a formalisation of combinatory logic, introduced by Feferman in the mid seventies [12] as base theory of his theories of explicit mathematics. These theories employ the logic of partial terms, which was introduced by Beeson in [1], [2]. Applicative theories corre-

Email address: eberhard@iam.unibe.ch (Sebastian Eberhard)

sponding to complexity classes have been introduced e.g. by Strahm [21, 22] for linear and polynomial time - and space classes, by Cantini for polynomial time [6], and by Kahle and Oitavem [16] for the polynomial hierarchy.

In contrast to corresponding theories of bounded arithmetic, applicative theories allow to prove the totality of functions of their complexity class without coding. This makes the lower bound proofs typically easier and more transparent. For an overview of weak applicative theories, we recommend Strahm's [23].

We present applicative theories for various logarithmic complexity classes. The theories are formulated over a base theory of words and contain induction principles to formalise the suitable forms of recursion. As Cantini's mentioned system they contain two predicates W and V, however their interpretation is slightly different. We have to be more restrictive about the permitted operations on V to achieve logarithmic strength: We cannot allow case distinction for elements of V of the following form.

$$\mathsf{case}(;y_1,y_2,y_3) := \begin{cases} y_2, & \text{if } mod_2(y_1)=0 \\ y_3, & \text{else} \end{cases}$$

Accordingly, the intended meaning of being an element of V differs from Cantini's theory, where elements of V just have a different role in the induction scheme. For the weaker theories presented in this article, $t \in \mathsf{V}$ informally means that t is a temporary inaccessible word, e.g. it can only be the input of functions not requiring to read off any of its bits. This property is fulfilled e.g. for the successor - and predecessor functions, and is designed to describe the role intermediate values[1] play during concatenation recursion. For the stronger theory of logspace strength, we allow at least to determine, whether a $t \in \mathsf{V}$ equals the empty word ϵ by the following case distinction given for safe inputs.

$$\mathsf{case}(;y_1,y_2,y_3) := \begin{cases} y_2, & \text{if } y_1=\epsilon \\ y_3, & \text{else} \end{cases}$$

If we compare our two-sorted theories with Cantini's, the main difference is that we do not only forbid elements of V to control recursion, but also during recursion restrict the way they can be used heavily.

[1]For $f(w,\vec{x};\vec{y})$ defined by some recursion scheme on notation on w, the intermediate values are $f(w',\vec{x};\vec{y})$ for $w' \subset w$. Analogously for functions not involving safe arguments.

Let us summarize the content of this paper. In section 2, we design applicative theories formalising concatenation recursion. We present three applicative theories of words in detail that correspond to implicit characterisations of complexity classes whose main principle is concatenation recursion or an extension thereof. Their provably total functions are the elements of the logarithmic hierarchy for LogT, alternating logarithmic time for AlogT, and polynomial time for PT.

The theories LogT, AlogT, and PT are introduced simultaneously, since they only differ very slightly in their induction schemes. Their induction formulas have two free variables, as in Kahle and Oitavem's [15, 16], which allows to express that $f(\mathsf{s}_i w, \vec{x})$ is a successor of $f(w, \vec{x})$ for a function f defined by concatenation recursion.

The lower bound of the theories LogT, AlogT, and PT is established by proving totality for all elements of corresponding word function algebras. Because of the computational completeness of the underlying combinatory algebra, we can directly produce terms representing these functions without any coding. Then, we prove by an easy induction that these terms represent total functions. In section 2.4, we prove the upper bound of the theories using a modification of Strahm's realisation approach [22]. This delivers an exact characterisation of LogT, AlogT and PT in terms of provably total functions.

In section 3, we present a new two-sorted algebra $\mathfrak{LS}$ for logarithmic space, which is just Cook and Bellantoni's B with a weakened case distinction and an additional initial function yielding the length of its input. In contrast to Bellantoni's description of logarithmic space as safe *unary* algebra, $\mathfrak{LS}$ contains also the fast growing members of logspace. The prize one has to pay is the addition of the initial function len having as output the length of its input. Implicit characterisations of logspace using a safe/normal distinction have also been developed by Neergaard [18], and Oitavem [19].

In section 4, we formalise $\mathfrak{LS}$ and Clote's algebra for logspace containing sharply bounded recursion [7], and obtain new two-sorted applicative theories of the same strength. These theories contain ordinary one-variable induction schemes.

2. Applicative theories for concatenation recursion

We introduce the theories LogT, AlogT, and PT in detail, and execute their proof-theoretic analysis.

2.1. Function algebras on words

For the logarithmic hierarchy, alternating logarithmic time, and polynomial time there exist well-known function algebras $\mathfrak{A}_1$, $\mathfrak{A}_2$ and $\mathfrak{A}_3$ on numbers. $\mathfrak{A}_1$ and $\mathfrak{A}_2$ were developed by Clote in [7]. $\mathfrak{A}_3$ was developed by Ishihara in [14]. However, we formulate our theories for words, and therefore it will be practical to work with function algebras *on words* of corresponding strengths.

Before defining these function algebras, we first explain some concepts. Words are given as elements of $\{0,1\}^*$, so they are finite sequences of zeros and ones. The length $|w| \in \mathbb{N}$ of a word w is defined in the obvious way. The word w consists of $|w|$ bits. Its first bit is the rightmost one and is given by $\mathsf{BIT}(0,w)$, the other bits are given by $\mathsf{BIT}(1,w)$, $\mathsf{BIT}(2,w), \cdots, \mathsf{BIT}(|w|-1,w)$. As usual, the most significant bits are at the left -, the least significant bits at the right side. The relation $\prec$ orders the words first by length, and if they have the same length lexicographically. For words w, v and numbers n, m we have $w \prec v$ exactly if

$$
\begin{aligned}
&|w| < |v| \vee (|w| = |v| \wedge \\
&(\exists n \leq |w|-1)(\forall m < n \leq |w|-1) \\
&(\mathsf{BIT}(m,w) = \mathsf{BIT}(m,v) \wedge \mathsf{BIT}(n,w) = 0 \wedge \mathsf{BIT}(n,v) = 1)
\end{aligned}
$$

We write $w \preceq v$ for $w \prec v \vee w = v$. We call this order the lexicographic order on words in the following. The existence of an order isomorphism I from $(\mathbb{W}, \preceq)$ to $(\mathbb{N}, \leq)$ allows us to give a bit function with two *words* as input.

Definition 1 $\mathsf{bit} : \mathbb{W}^2 \to \mathbb{W}$ *is given as the function fulfilling the following specifications.*

- *If $I(w)$ is smaller than the length of v, $\mathsf{bit}(w,v)$ equals the $I(w)$-th bit of v in the sense of the* BIT *function.*
- *In all other cases, $\mathsf{bit}(w,v)$ is 0.*

Next, we define a length function on words giving a word as output, relying on I.

Definition 2 $\mathsf{len} : \mathbb{W} \to \mathbb{W}$ *is defined such that $\mathsf{len}(w) = v$ exactly if the length of w is $n \in \mathbb{N}$ and $I^{-1}(n) = v$.*

Definition 3 *The function algebras $\mathfrak{W}_1$, $\mathfrak{W}_2$ and $\mathfrak{W}_3$ on words (corresponding to $\mathfrak{A}_1$, $\mathfrak{A}_2$ and $\mathfrak{A}_3$) contain the following initial functions.*

- *the constant empty word function.*

- *the word successor functions* s_0, s_1, *concatenating 0 or 1, respectively, at the right side of their input.*
- *projection functions of arbitrary arity.*
- *the function* bit.
- *the function* len.
- *the function* $\times$ *where* $w \times v$ *is the length of the* v *fold concatenation of* w *with itself.*
- *the function* e *where* $\mathsf{e}(w)$ *is the word* w *without its leading zeros, e.g. zeros bits at the left of any one bit.*

The algebras are closed under various operations.

- *The function algebras* $\mathfrak{W}_1$, $\mathfrak{W}_2$ *and* $\mathfrak{W}_3$ *are closed under composition.*
- *The function algebras* $\mathfrak{W}_1$ *and* $\mathfrak{W}_2$ *are closed under the following scheme of concatenation recursion on notation* CRN. *The notation* $\mathsf{s}_{\mathsf{bit}(\cdots)}$ *abbreviates in the following always a case distinction on the value of* $\mathsf{bit}(\cdots)$.

$$\begin{aligned} f(\epsilon, \vec{y}) &= g(\vec{y}) \\ f(\mathsf{s}_0(x), \vec{y}) &= \mathsf{s}_{\mathsf{bit}(\epsilon, h_0(x,\vec{y}))}(f(x, \vec{y})) \\ f(\mathsf{s}_1(x), \vec{y}) &= \mathsf{s}_{\mathsf{bit}(\epsilon, h_1(x,\vec{y}))}(f(x, \vec{y})) \end{aligned}$$

- *The function algebra* $\mathfrak{W}_2$ *is closed under the following scheme of* k*-bounded recursion* $k - BRN$ *for each* $k \in \mathbb{W}$. *We write* $w \leq v$ *for* $w, v \in \mathbb{W}$ *exactly if* v *is at least as long as* w.

$$\begin{aligned} f(\epsilon, \vec{y}) &= g(\vec{y}) \mid k \\ f(\mathsf{s}_0(x), \vec{y}) &= h_0(x, \vec{y}, f(x, \vec{y}))) \mid k \ , \\ f(\mathsf{s}_1(x), \vec{y}) &= h_1(x, \vec{y}, f(x, \vec{y}))) \mid k \end{aligned}$$

where

$$x|y = \begin{cases} x, & \text{if } x \leq y \\ y, & \text{else} \end{cases}$$

- *The function algebra $\mathfrak{W}_3$ is closed under the following scheme of extended concatenation recursion on notation CRN^+.*

$$\begin{aligned} f(\epsilon, \vec{y}) &= g(\vec{y}) \\ f(\mathsf{s}_0(x), \vec{y}) &= \mathsf{s}_{\mathsf{bit}(\epsilon, h_0(x,\vec{y},f(x,\vec{y})))}(f(x,\vec{y})) \\ f(\mathsf{s}_1(x), \vec{y}) &= \mathsf{s}_{\mathsf{bit}(\epsilon, h_1(x,\vec{y},f(x,\vec{y})))}(f(x,\vec{y})) \end{aligned}$$

For concatenation recursion on notation, if h_0 and h_1 give only 0 or 1 as output, we drop bit.

We have to include the eraser function because the scheme of concatenation recursion is weaker for word algebras than for number algebras. This is because of the different properties of the successor functions: within word algebras, they always increase their input, whereas $\mathsf{s}_0(0) = 0$ holds in the setting of numbers. This makes the usual definition of sharply bounded quantification impossible in the setting of words. For all function algebras on words we present in this paper, the addition of the eraser can be easily proved to be necessary: For any function F contained in one of these algebras, the number of lengths an output $F(\vec{x})$ can possibly have for inputs $\vec{x}$ of fixed length can be shown to be too small to include the eraser.

Executing a similar bootstrapping process as in Clote's [7] for the word algebras (containing the eraser), it is proved that they exactly contain word functions of their corresponding complexity class.

2.2. The systems LogT, AlogT *and* PT

The above mentioned theories differ only minimally, so we develop them simultaneously. We make use of a safe predicate V to express that for F defined by concatenation recursion, the recursion step function must not depend on the intermediate values of F. The systems are based on an applicative base theory including the axioms for a combinatory algebra and basic predicates W, V which are interpreted as the set $\mathbb{W} = \{0,1\}^*$ of binary words in the standard interpretation.

2.2.1. The applicative language L

Our basic language L is a first order language for the logic of partial terms which includes:

- variables $a, b, c, x, y, z, u, v, f, g, h, \ldots$
- the applicative constants k, s, p, p_0, p_1, d_W, ϵ, s_0, s_1, p_W, s_ℓ, p_ℓ, $\mathsf{c}_\subseteq$, $*$, $\times$, len, bit, e.

- relation symbols $=$ (equality), $\downarrow$ (definedness), W (binary words)
- arbitrary term application $\circ$

The terms $(r, s, t, \dots)$ and formulas $(A, B, C, \dots)$ of L are defined using the connectives $\neg, \wedge, \vee, \rightarrow$ and the quantifiers $\exists, \forall$. We assume the following standard abbreviations and syntactical conventions:

$$\begin{array}{rcl} t_1 t_2 \dots t_n & := & (\dots (t_1 \circ t_2) \circ \dots \circ t_n) \\ s(t_1, \dots, t_n) & := & s t_1 \dots t_n \\ t_1 \simeq t_2 & := & t_1{\downarrow} \vee t_2{\downarrow} \rightarrow t_1 = t_2 \\ t \in \mathsf{W} & := & \mathsf{W}(t) \\ t : \mathsf{W}^k \rightarrow \mathsf{W} & := & (\forall x_1 \dots x_k \in \mathsf{W}) t x_1 \dots x_k \in \mathsf{W} \\ s \leq t & := & \mathsf{c}_{\subseteq}(1{\times}s, 1{\times}t) = 0 \\ s \leq_{\mathsf{W}} t & := & s \leq t \wedge s \in \mathsf{W} \end{array}$$

In the following we often write $A[\vec{x}]$ in order to indicate that the variables $\vec{x} = x_1, \dots, x_n$ may occur freely in A. We write $A[\vec{s}]$ for the substitution of the free variables of A by $\vec{s}$.

For vectors $\vec{y}$ of any domain, we use y_i to denote the i-th component of $\vec{y}$ starting with $i = 1$. Finally, let us write $\overline{w}$ for the canonical closed L term denoting the binary word $w \in \mathbb{W}$.

2.2.2. Rules and axioms of LogT, AlogT *and* PT

We use a base theory B' that is very similar to Strahm's theory B introduced in [21, 22].

The logic of B' is the classical logic of partial terms due to Beeson [1, 2] [2]. The non-logical axioms of B' include axioms of a partial combinatory algebra:

$$\mathsf{k}xy = x, \qquad \mathsf{s}xy{\downarrow} \wedge \mathsf{s}xyz \simeq xz(yz)$$

We also have the usual axioms for pairing p with projections p_0 and p_1. Then, we add axioms stating that the further applicative constants, representing simple functions on words in the standard model, fulfil the expected recursion equations on W. These axioms do not contain the predicate V and have the following form.

[2] The reason for working with a partial instead of a total version of B' is that we do not want to exclude the very natural recursion theoretic standard model of the introduced theories having the words as universe and interpreting application as partial recursive function application in the sense of ordinary recursion theory. Nevertheless, all upper bound proofs will be given for total versions of the introduced systems.

- defining axioms for the binary words W with ϵ, the successors s_0, s_1 which concatenate 0, 1, respectively, at the right side of a word, and the predecessor p_W which deletes the least significant bit.
- defining axioms for $\mathsf{c}_\subseteq$ which represents the initial subword relation.
- defining axioms for s_ℓ, p_ℓ which yield the lexicographic successor or -predecessor, respectively.
- definition by cases d_W on W, given as follows for variables x, y, u, v:
  - $x \in \mathsf{W} \wedge y \in \mathsf{W} \wedge x = y \to \mathsf{d}_\mathsf{W} uvxy = u$
  - $x \in \mathsf{W} \wedge y \in \mathsf{W} \wedge x \neq y \to \mathsf{d}_\mathsf{W} uvxy = v$
- word concatenation $*$, word multiplication $\times$

These axioms are fully spelled out in [21, 22]. We have the following axioms for the new constants.

(len.1) $\mathsf{len} : \mathsf{W} \to \mathsf{W}$
(len.2) $\mathsf{len}\epsilon = \epsilon$
(len.2) $x \in \mathsf{W} \to \mathsf{len}(\mathsf{s}_i x) = \mathsf{s}_\ell(\mathsf{len}x)$
(bit.1) $\mathsf{bit} : \mathsf{W}^2 \to \mathsf{W}$
(bit.2) $x \in \mathsf{W} \to \mathsf{bit}(\epsilon, x) = \mathsf{d}_\mathsf{W}(1, 0, \mathsf{s}_1(\mathsf{p}_\mathsf{W} x), x)$
(bit.3) $x \in \mathsf{W} \to \mathsf{bit}(x, \epsilon) = 0$
(bit.4) $x, y \in \mathsf{W} \to \mathsf{bit}(x, y) = \mathsf{bit}(\mathsf{s}_\ell x, \mathsf{s}_i y)$
(e.1) $\mathsf{e} : \mathsf{W} \to \mathsf{W}$
(e.2) $\mathsf{e}\epsilon = \epsilon$
(e.3) $x \in \mathsf{W} \to \mathsf{e}(\mathsf{s}_0 x) = \mathsf{d}_\mathsf{W}(\epsilon, \mathsf{s}_0(\mathsf{e}x), \mathsf{e}x, \epsilon)$
(e.4) $x \in \mathsf{W} \to \mathsf{e}(\mathsf{s}_1 x) = \mathsf{s}_1(\mathsf{e}x)$

Since len yields the length of words, we often write $|t|$ instead of $\mathsf{len} \circ t$.

To motivate the axioms and rules for V, we have to give the informal meaning of the predicates W and V. As we mentioned already, $v \in \mathsf{V}$ is intended to mean that v is a stored, temporary inaccessible word while W contains fully accessible words. Let us explain these ideas in more detail. We can sensitively apply any of our initial functions to $w \in \mathsf{W}$, especially, we can calculate its bits, i.e. we can fully access w. So, $w \in \mathsf{W}$ is given to us similarly as content stored on a usual read-write tape of a Turing machine. On the other hand, to $v \in \mathsf{V}$ we allow only the application of the successor

and predecessor functions. The motivation behind this is that given a word $v \in \mathbb{W}$ its successors and predecessor can be produced without knowing any of its bits. The knowledge where the word ends is already sufficient. Content in V is given to us similarly as content stored on a write only tape to a Turing machine having the write head always on the rightmost bit of the word it contains. Content in V can be bitwise extended or deleted but not accessed.

Sequents of the form $s \in \mathsf{V} \to t \in \mathsf{V}$ are interpreted as claiming that a transformation of content s into content t is possible where both are temporary inaccessible. Content in V is not inaccessible forever: if it is possible to produce temporary inaccessibly stored content without assuming anything about other temporary inaccessible content, we can transfer it into stable fully accessible content. This corresponds to the idea that at the end of a computation process, we can read off the result, even if during the computation there is no time or space to do so.

To formalise concatenation recursion, we let the intermediate values $f(x, \vec{y})$ be elements of V. In this way, we express that the induction step functions do not depend on them. Only at the end of computation, we dispose of $f(x, \vec{y})$. According to our motivation, we give the following axioms.

(V-intro)	$x \in \mathsf{W} \to x \in \mathsf{V}$
(V-ext)	$x \in \mathsf{V} \to \mathsf{s}_i x \in \mathsf{V}$
(V-del)	$x \in \mathsf{V} \to \mathsf{p}_\mathsf{W} x \in \mathsf{V}$

To define the rule which allows to replace V- by W-occurrences, we need the concept of a positive L formula.

Definition 4 *For s, t being* L *terms, an* L *formula A is positive if A is build from formulas of the form $s = t$, $s \simeq t$, $s{\downarrow}$, $s \in \mathsf{W}$, $s \in \mathsf{V}$, using the connectives $\wedge, \vee$ and the quantifiers $\forall, \exists$.*

Now, let A be an arbitrary formula not containing V, and B_W being the positive formula B with all occurrences of V replaced by W.

(V-elim)
$$\frac{A \to B}{A \to B_\mathsf{W}}$$

This concludes the description of the axioms available for V. Note that we cannot even check for elements in V whether they equal ϵ. We will allow this later to construct systems of logspace strength.

The induction scheme formalising concatenation recursion has to be formulated such that for an induction formula $A[x, y] \equiv fx = y$ and for a function

f defined by concatenation recursion, y is stored but temporary inaccessible. In contrast, the extended concatenation recursion is formalised by a scheme that allows full access to such a y. This results in the following induction schemes for PT and AlogT, respectively, where $A[x, y]$ is a positive formula without any occurrences of W or V.

$$\begin{aligned}(\mathsf{PT}\text{-Ind}) \qquad & (\exists y \in \mathsf{W})A[\epsilon, y] \wedge \\ & (\forall x, y \in \mathsf{W})\big(A[x, y] \to A[\mathsf{s}_i x, \mathsf{s}_0 y] \vee A[\mathsf{s}_i x, \mathsf{s}_1 y]\big) \to \\ & (\forall x \in \mathsf{W})(\exists y \in \mathsf{W})A[x, y]\end{aligned}$$

$$\begin{aligned}(\mathsf{AlogT}\text{-Ind}) \qquad & (\exists y \in \mathsf{V})A[\epsilon, y]) \wedge \\ & (\forall x \in \mathsf{W})(\forall y \in \mathsf{V})\big(A[x, y] \to A[\mathsf{s}_i x, \mathsf{s}_0 y] \vee A[\mathsf{s}_i x, \mathsf{s}_1 y]\big) \to \\ & (\forall x \in \mathsf{W})(\exists y \in \mathsf{V})A[x, y]\end{aligned}$$

The scheme for AlogT is weaker than that of PT because we have to prove the induction step for inaccessible words y. Finally, the induction scheme for LogT restricts the scheme of AlogT by allowing only positive induction formulas A which are W, V and *disjunction* free, except of disjunctions occurring within formulas of the form $s \simeq t$. In the following, we drop the V-axioms from PT since they are unnecessary. This concludes the description of the theories LogT, AlogT, and PT.

The standard open term model $\mathcal{M}(\lambda\eta)$ for B can be easily generalised to model the introduced theories: Take the universe of open λ terms and consider the usual reduction of the extensional untyped lambda calculus $\lambda\eta$, augmented by suitable reduction rules for the constants other than k and s. Interpret application as juxtaposition. Two terms are equal if they have a common reduct, W and V denote the set of terms which reduce to a "standard" word $\overline{w}$.

2.3. Lower bound

We find a lower bound for the theories LogT, AlogT and PT in the sense of provably total functions. We use the following standard definition.

Definition 5 *A function $F : \mathbb{W}^n \to \mathbb{W}$ is called provably total in an $\mathcal{L}$ theory Th, if there exists a closed* L *term t_F such that*

(i) $\mathsf{Th} \vdash t_F : \mathsf{W}^n \to \mathsf{W}$ *and, in addition,*

(ii) $\mathcal{M}(\lambda\eta) \vDash t_F \overline{w}_1 \cdots \overline{w}_n = \overline{F(w_1, \ldots, w_n)}$ *for all $w_1, \ldots, w_n$ in $\mathbb{W}$.*

Lemma 6

- *All functions of* $\mathfrak{W}_1$ *are provably total in* LogT.
- *All functions of* $\mathfrak{W}_2$ *are provably total in* AlogT.
- *All functions of* $\mathfrak{W}_3$ *are provably total in* PT.

Proof.

The initial functions are clearly provably total and the provably total functions are closed under composition for all introduced theories. Now, let us deal with the case where the function $F(x, \vec{z})$ is defined by concatenation recursion applied to $G(\vec{z})$ and $H_i(x, \vec{z})$ being both in $\mathfrak{W}_1$ or $\mathfrak{W}_2$, respectively. The induction hypothesis delivers terms t_G and t_{H_i} representing G and H_i. We find a closed term t_F such that

$$\begin{aligned} t_F\epsilon\vec{z} &\simeq t_G\vec{z} \\ t_F(\mathsf{s}_i w)\vec{z} &\simeq \mathsf{d_W}(\mathsf{s}_0(t_F w\vec{z}), \mathsf{s}_1(t_F w\vec{z}), \mathsf{bit}(\epsilon, t_{H_i} w\vec{z}), 0) \end{aligned}$$

We take $t_F x\vec{z} = y$ as induction formula $A[x, y]$ and assume that $\vec{z} \in \mathsf{W}$. Let us show that the premisses of (LogT-Ind) or (AlogT-Ind), respectively, hold. The first conjunct holds because of the induction hypothesis for G and because $\mathsf{W} \subseteq \mathsf{V}$. To prove the second conjunct, we can assume $t_F x\vec{z} = y \in \mathsf{V}$ for $x \in \mathsf{W}$. Because of the properties of the H_i, we have that $\mathsf{bit}(\epsilon, t_{H_i} x\vec{z})$ equals 0 or 1. $y \in \mathsf{V}$ implies $\mathsf{s}_i y \in \mathsf{V}$, which means that all components of the definition by cases that is partially equal to $t_F(\mathsf{s}_i x)\vec{z}$ are defined. If we now assume that $\mathsf{bit}(\epsilon, t_{H_i} w\vec{z})$ equals 0, we derive $t_F \mathsf{s}_i x\vec{z} = \mathsf{s}_0 y$. The other case works analogously. The application of induction delivers $(\forall x \in \mathsf{W})(\exists y \in \mathsf{V}) t_F x = y$, which yields the totality of t_F using (V-elim).

For a function F defined by extended concatenation recursion, we define the term t_F similarly as before. The crucial difference is that the step functions H_i now depend on the intermediate values of F represented by y. To be able to sensibly apply t_{H_i} to y we have to impose $y \in \mathsf{W}$. Let us mention that the lower bound proof of PT also works if the predicate V is dropped from the theory.

Now, let us deal with the case where the function $F(x, \vec{y})$ is defined by w-bounded recursion applied to $G(\vec{y})$ and $H_i(x, \vec{y})$ both being in $\mathfrak{W}_2$. The induction hypothesis delivers terms t_G and t_{H_i} representing G, H_i. We find a closed term t_F such that

$$\begin{aligned} t_F\epsilon\vec{z} &\simeq t_G\vec{z} \mid \overline{w} \\ t_F(\mathsf{s}_i w)\vec{z} &\simeq t_{H_i} w(t_F w\vec{z})\vec{z} \mid \overline{w} \end{aligned}$$

We define the formula $s \preccurlyeq \overline{w}$ for each $w \in \mathbb{W}$ to be a disjunction composed of all disjuncts of the form $s = \overline{v}$ where $v \leq w$ (assume an arbitrary but fixed bracketing). In the base theory B' (see page 7) we can prove the following by external induction for all $w \in \mathbb{W}$.

$$\mathsf{B}' \vdash s \preccurlyeq \overline{w} \leftrightarrow s \leq_{\mathsf{W}} \overline{w}$$

To prove the totality of t_F, we take $t_F x\vec{z} \preccurlyeq \overline{w}$ as induction formula and assume $\vec{z} \in \mathsf{W}$. Then, the first conjunct of the antecedent holds because of the induction hypothesis for G. To prove the second conjunct, we can assume $t_F x\vec{z} \preccurlyeq \overline{w}$ for $x \in \mathsf{W}$. This implies that $t_F x\vec{z}$ is a word, which yields together with the induction hypothesis for the H_i the desired result. Note, that the axiom $x \in \mathsf{V} \to \mathsf{p}_{\mathsf{W}} x \in \mathsf{V}$ was not needed to prove the lower bound. □

2.4. Upper bound

For the upper bound proof, we work with total versions $\mathsf{LogT}\!\downarrow$, $\mathsf{AlogT}\!\downarrow$ and $\mathsf{PT}\!\downarrow$ of the introduced theories. They are obtained from the partial theories by the following modifications:

- We drop $\downarrow$, and replace all formulas of the form $s \simeq t$ by $s = t$ in the axioms and rules.
- We work with the usual classical logic.
- We redefine the notion of positive formulas as follows.

Definition 7 *For s, t being* L *terms, an* L *formula A is positive if A is build from formulas of the form $s = t$, $s \in \mathsf{W}$, $s \in \mathsf{V}$, using the connectives $\wedge, \vee$ and the quantifiers $\forall, \exists$.*

It is easy to see that $\mathsf{LogT}\!\downarrow$, $\mathsf{AlogT}\!\downarrow$ and $\mathsf{PT}\!\downarrow$ are equivalent to the extensions of LogT, AlogT and PT by the axiom $(\forall x, y)xy\!\downarrow$.

We formulate $\mathsf{LogT}\!\downarrow$, $\mathsf{AlogT}\!\downarrow$ and $\mathsf{PT}\!\downarrow$, which we just call LogT, AlogT and PT in the following, in Gentzen's classical sequent calculus LK. We assume familiarity with LK as it is presented, for example, in Girard's [13]. By formulating induction as a rule, we obtain systems with only positive main formulas for all theories. For the sequent style systems, we give the V-elimination rule as follows for V-free Γ, Δ, a positive B, and B_{W} being B with all occurrences of V replaced by W.

$$\frac{\Gamma \Rightarrow B, \Delta}{\Gamma \Rightarrow B_{\mathsf{W}}, \Delta}$$

Note that compared to Cantini's [6], we allow more general side formulas, and B can be a positive formula of arbitrary complexity. The reason why this is possible is that for our realisation approach an elimination of non-positive cuts is sufficient.

By standard techniques, see e.g. Girard's [13], we can eliminate all cuts with non-positive cut-formula. This implies that sequents containing only positive formulas have proofs containing only positive formulas. The above sketched transformation of the theories into sequent style, and the subsequent partial cut-elimination are described in detail in Strahm's [22] on pages 24-26 for similar theories.

Due to the cut-elimination result, we can restrict the realisation to positive formulas. We use two realisation relations $\mathfrak{r}$ and $\mathfrak{R}$. The first trivializes the V-predicate, the second treats V exactly like W. Correspondingly, we deliver for provable sequents two realisation functions: F operating on $\mathfrak{r}$ realisers, and $F^{\Box}$ on $\mathfrak{R}$ realisers. The interplay of F and $F^{\Box}$ allows to catch both aspects of V: On the one hand the possibilities to derive something from $t \in \mathsf{V}$ are very restricted, so we are not allowed to use its realiser as freely as a realiser of $t \in \mathsf{W}$; on the other hand because of (V-elim) under some conditions we have to produce a full W-atom (e.g. a formula of the form $t \in \mathsf{W}$) realiser from a V-atom realiser.

Let us define both realisation relations. We let $\langle \cdot, \cdot \rangle$ denote the word analogue of the pairing function defined by Clote in [7] which is in the logarithmic hierarchy and has roughly the same growth rate as concatenation. We use pairing functions with higher arities in the analogous sense. We write $\rho_0, \rho_1, \cdots$ for the components of a word ρ relative to this pairing supposing that ρ is a pair.

Definition 8 (Realisation relation $\mathfrak{r}$)

$\rho \ \mathfrak{r} \ \mathsf{W}(t)$	*iff*	$\mathcal{M}(\lambda\eta) \vDash t = \overline{\rho}$,
$\rho \ \mathfrak{r} \ \mathsf{V}(t)$	*iff*	$\rho = \epsilon$,
$\rho \ \mathfrak{r} \ (t_1 = t_2)$	*iff*	$\rho = \epsilon$ *and* $\mathcal{M}(\lambda\eta) \models t_1 = t_2$,
$\rho \ \mathfrak{r} \ (A \wedge B)$	*iff*	$\rho = \langle \rho_0, \rho_1 \rangle$ *and* $\rho_0 \ \mathfrak{r} \ A$ *and* $\rho_1 \ \mathfrak{r} \ B$
$\rho \ \mathfrak{r} \ (A \vee B)$	*iff*	$\rho = \langle i, \rho_1 \rangle$ *and either* $i = 0$ *and* $\rho_1 \ \mathfrak{r} \ A$ *or* $i = 1$ *and* $\rho_1 \ \mathfrak{r} \ B$,
$\rho \ \mathfrak{r} \ (\forall x) A(x)$	*iff*	$\rho \ \mathfrak{r} \ A(u)$ *for a fresh variable* u,
$\rho \ \mathfrak{r} \ (\exists x) A(x)$	*iff*	$\rho \ \mathfrak{r} \ A(t)$ *for some term* t.

Definition 9 (Realisation relation $\mathfrak{R}$) *The realisation relation $\mathfrak{R}$ is defined as $\mathfrak{r}$ except that the clause*

$$\rho \ \mathfrak{r} \ \mathsf{V}(t) \qquad \textit{iff} \qquad \rho = \epsilon$$

is replaced by

$$\rho \ \mathfrak{R} \ \mathsf{V}(t) \qquad \textit{iff} \qquad \mathcal{M}(\lambda\eta) \models t = \overline{\rho}$$

Sequences Γ of formulas are realised as usual by tuples $\vec{\rho}$ of realisers where the i-th component of $\vec{\rho}$ realises the i-th formula of Γ.

Note that for a formula A realised by ρ relative to $\mathfrak{r}$ or $\mathfrak{R}$, we can talk about the atoms of A that are realised by ρ in a natural way. This is so because ρ just contains individual realisers of possibly substituted atoms of A (with multiplicity), within a structure of pairs allowing to find for each individual realiser the corresponding atom. Let A be e.g. the formula

$$(\exists x)(x \in \mathsf{W} \wedge (x = 0 \vee x = 0)).$$

Its $\mathfrak{r}$ -realiser $\langle 0, \langle 0, \epsilon\rangle\rangle$ realises the first and the second atom from the left, but does not realise the third one.

For any positive formula A, there is a function $\cdot^A$ (projection function) within the logarithmic hierarchy that transfers a given $\mathfrak{R}$ realisers of A into a $\mathfrak{r}$ realiser of A, just by inserting ϵ at the suitable positions. We can assume the following properties for this function.

- $\cdot^{A[\vec{s}]}$ and $\cdot^{A[\vec{t}]}$ denote the same function.
- ρ and ρ^A realise the same atoms in the sense described above.
- $\cdot^{QxA[x]}$ is given as $\cdot^{A[u]}$ for $Q = \exists, \forall$ and a fresh variable u.

The projection function can easily be generalised to tuples of realisers and sequences of formulas, and is written as $\cdot^\Gamma$ in such cases. We write just $\cdot^*$ for projection functions, if Γ is clear from the context.

Theorem 10 *Let T_0, T_1, T_2 denote LogT, AlogT, PT, respectively. Let $\Gamma \Rightarrow \Delta$ be a sequent of positive formulas with $\Gamma \equiv A_1, \ldots, A_n$ and $\Delta \equiv D_1, \ldots, D_m$ and assume $\mathsf{T}_i \vdash^\star \Gamma[\vec{u}] \Rightarrow \Delta[\vec{u}]$. Then there exists functions $F, F^\Box : \mathbb{W}^n \to \mathbb{W}$ in $\mathfrak{W}_i$ such that for each substitution $[\vec{s}]$ and each $\vec{\rho}\ \mathfrak{R}\ \Gamma[\vec{s}]$ the following conditions hold.*

- $F^{\square}(\vec{\rho})\ \mathfrak{R}\ \Delta[\vec{s}]$
- $F(\vec{\rho}\,^{\Gamma})_0 = F^{\square}(\vec{\rho})_0 = k$
- $F(\vec{\rho}\,^{\Gamma})_1 = F^{\square}(\vec{\rho})_1^{D_k}$

Let us explain the three conditions. The first conditions claims that a function $F^{\square}$ delivers an $\mathfrak{R}$ realiser of $\Delta[\vec{s}]$, which means a realiser reflecting the V-predicate, on input $\vec{\rho}$. The second condition claims that the realisation functions F and $F^{\square}$ pick the same formula $D_k[\vec{s}]$ of $\Delta[\vec{s}]$ to realise. Note, that the input of the realisation function F is a projection of the input of $F^{\square}$. Finally, the third condition connects the realisers delivered by F and $F^{\square}$, stating that the first is a projection of the second.

Proof. We proof the theorem by induction on the depth of the positive proof. Note that for PT, we only have to give a realisation function F. The logical - and applicative axioms are realised easily. Also the logical rules do not pose special difficulties.

Let us realise the V-elimination rule given as follows where V does not occur in Γ, and B_{W} is B with all occurrences of V replaced by W.

$$\frac{\Gamma \Rightarrow B, \Delta}{\Gamma \Rightarrow B_{\mathsf{W}}, \Delta}$$

For the premise, we have realisation functions $P, P^{\square}$. We construct the realisation function F. Let $\vec{\rho}$ be the given realisers for Γ relative to realisation relation $\mathfrak{r}$. Since Γ does not contain V they are realisers of Γ relative to $\mathfrak{R}$. An application of $P^{\square}$ delivers an $\mathfrak{R}$ realiser of B, Δ. Because $\mathfrak{R}$ realisers of $t \in \mathsf{V}$ are equal to $\mathfrak{r}$ realisers of $t \in \mathsf{W}$, $P^{\square}(\vec{\rho})$ simultaneously yields an $\mathfrak{r}$ and $\mathfrak{R}$ realiser of B_{W}, Δ. Therefore, we define F and $F^{\square}$ as $P^{\square}$.

Let us now realise the induction scheme of AlogT with premisses

- $\Gamma \Rightarrow (\exists y \in \mathsf{V})A[\epsilon, y], \Delta$
- $\Gamma, x \in \mathsf{W}, y \in \mathsf{V}, A[x, y] \Rightarrow A[\mathsf{s}_0 x, \mathsf{s}_0 y] \vee A[\mathsf{s}_0 x, \mathsf{s}_1 y], \Delta$
- $\Gamma, x \in \mathsf{W}, y \in \mathsf{V}, A[x, y] \Rightarrow A[\mathsf{s}_1 x, \mathsf{s}_0 y] \vee A[\mathsf{s}_1 x, \mathsf{s}_1 y], \Delta$

and the conclusion

- $\Gamma, t \in \mathsf{W} \Rightarrow (\exists y \in \mathsf{V})A[t, y], \Delta,$

where the usual variable condition applies. We assume realisation functions $G, G^{\square}, H_0, H_0^{\square}, H_1, H_1^{\square}$ for the premisses and that Γ contains n formulas.

First, we define a function $I : \mathbb{W}^{n+1} \to \mathbb{W}$ which produces $\mathfrak{r}$-realisers of the induction formula A relative to specific substitutions. This can be done by the following c_A-bounded recursion for a $c_A \in \mathbb{W}$ such that for all $\rho \in \mathbb{W}$ and all terms s, t

$$\rho \ \mathfrak{r} \ A[s,t] \Rightarrow \rho \leq c_A.$$

The existence of c_A easily follows from the following lemma which can be proved by induction on the complexity of the formula A.

Lemma 11 *Let A be a positive, W-free formula. Then there exists a word c_A such that for any substitution $[\vec{s}]$ and any word ρ*

$$\rho \ \mathfrak{r} \ A[\vec{s}] \Rightarrow \rho \leq c_A.$$

We abbreviate multiple projections of a word w as $w_{n_0,\cdots,n_m}$ and define I as follows.

$$\begin{aligned} I(\vec{z},\epsilon) &= G(\vec{z})_{1,1} \mid c_A \\ I(\vec{z},\mathsf{s}_i w) &= H_i(\vec{z},w,\epsilon,I(\vec{z},w))_{1,1} \mid c_A \end{aligned}$$

Under the assumption $\vec{z} \ \mathfrak{r} \ \Gamma$, for any $w \in \mathbb{W}$, $I(\vec{z},w)$ delivers a realiser of $A[\overline{w},\overline{v}]$ for some $v \in \mathbb{W}$, as long as no side formula is realised.

Let us define a second auxiliary function $Q : \mathbb{W}^{n+1} \to \mathbb{W}$ producing an $\mathfrak{R}$ realisers for the first inner conjunct of $(\exists y \in \mathsf{V})A[t,y]$ given an $\mathfrak{R}$-realiser $\vec{z}$ of Γ, presupposed that no side formula is realised.

$$\begin{aligned} Q(\vec{z},\epsilon) &= G^{\Box}(\vec{z})_{1,0} \\ Q(\vec{z},\mathsf{s}_i w) &= \mathsf{s}_{\mathsf{bit}(\epsilon,H_i[(\vec{z})^*,w,\epsilon,I((\vec{z})^*,w)]_{1,0})} Q(\vec{z},w) \end{aligned}$$

Note, that we have to use H_i instead of $H_i^{\Box}$ to find the suitable successor because we are not allowed to replace ϵ by a term containing intermediate values of Q. Because of the induction hypothesis for the premise realisation functions, both auxiliary functions are in $\mathfrak{W}_2$.

To define the realisation function F, we have to decide whether a side formula is realised. This case distinction cannot be integrated into the recursive definition of the realisation function as usual, because of the weakness of CRN. Therefore, we have to distinguish cases independently of earlier values of F. Let us define the realisation function F as follows:

Case 1 $G(\vec{z})_0 \neq 1$. We define $F(\vec{z},w)$ as $G(\vec{z})$.

Case 2 $G(\vec{z})_0 = 1 \wedge (\exists l \preccurlyeq |w|)P(l, w, \vec{z})$ where

$$P(l, w, \vec{z}) :\Leftrightarrow (\exists i)(\mathsf{s}_i(\mathsf{msp}(l, w)) = \mathsf{msp}(l, w) \wedge H_i(\vec{z}, \mathsf{msp}(l, w), \epsilon, I(\vec{z}, \mathsf{msp}(l, w)))_0 \neq 1)$$

msp denotes the most significant part function [3]. This case applies if in the course of recursion, we hit a side formula. We define $F(\vec{z}, w)$ as

$$H_i(\vec{z}, \mathsf{msp}(j, w), \epsilon, I(\vec{z}, \mathsf{msp}(j, w))),$$

where $j = (\nu y \preceq |w|)P(y, w, \vec{z})$, with ν being the usual maximal witness operator [4].

Case 3 Case 1 and 2 are not satisfied. We define $F(\vec{z}, w)$ as $\langle 1, \langle \epsilon, I(\vec{z}, w)\rangle\rangle$.

We can define the realisation function $F^{\Box}$ very similarly as F, using the same case distinction. The main difference is that in the third case we have to produce a non trivial realiser of the V-occurrence of the induction formula. We use in the following the same abbreviations as above.

Case 1 $G((\vec{z})^*)_0 \neq 1$. We define $F^{\Box}(\vec{z}, w)$ as $G^{\Box}(\vec{z})$.

Case 2 $G((\vec{z})^*)_0 = 1 \wedge (\exists l \preccurlyeq |w|)P(l, w, \vec{z})$. We define $F^{\Box}(\vec{z}, w)$ as

$$H_i^{\Box}(\vec{z}, \mathsf{msp}(j, w), Q(\vec{z}, \mathsf{msp}(j, w)), I((\vec{z})^*, \mathsf{msp}(j, w))),$$

where $j = (\nu y \preceq |w|)P(y, w, \vec{z})$, with ν being again the usual maximal witness operator.

Case 3 Case 1 and 2 are not satisfied. We define $F^{\Box}(\vec{z}, w)$ as

$$\langle 1, \langle Q(\vec{z}, w), I((\vec{z})^*, w)\rangle\rangle.$$

[3] The most significant part function msp is given in Clote's [7] on page 20 for number inputs. $\mathsf{msp}(n, m)$ outputs the leftmost n bits of m. Using the isomorphism between $(\mathbb{W}, \preceq)$ and $(\mathbb{N}, \leq)$ this function is transfered to word inputs, analogously as bit and len on page 4.

[4] All complexity classes analysed in this article are closed under the operator ν with inputs $w \in \mathbb{W}$ and $\vec{v} \in \mathbb{W}^n$ for any $n \in \mathbb{N}$ such that for any function $f : \mathbb{W}^{n+1} \to \mathbb{W}$ $\nu(w, \vec{v}, f)$ outputs the maximal $w' \preceq |w|$ relative to $\preceq$ such that $f(w', \vec{v}) = 0$. See Clote's [7] for details. In our article, we insert as f a relation P on $\mathbb{W}^{n+1}$ and write $(\nu y \preceq |w|)P(y, \vec{v})$ for $\nu(w, \vec{v}, f)$.

$\mathfrak{W}_1$ is closed under sharply bounded quantification and the sharply bounded minimal witness - and maximal witness operator (see Clote's [7]), which implies that the functions F, $F^{\square}$ are in $\mathfrak{W}_2$. We prove their correctness by an easy external induction on w. We use that the case distinctions for F and $F^{\square}$ exactly correspond to each other.

To deal with the weaker induction scheme of LogT, we argue similarly. Since the induction formula does not contain disjunctions this time, we can assume that it is always realised by the same word c_A. Therefore, we get correct realisation functions from the functions F, $F^{\square}$ above by replacing all terms of the form $I(a,b)$ by c_A. Since I is not needed, the modified realisation functions are in $\mathfrak{W}_1$.

To deal with the induction scheme of PT, we define the realisation function F by bounded recursion. We abbreviate $H_i(\vec{z},w,F(\vec{z},w)_{1,0},F(\vec{z},w)_{1,1})$ as $\tilde{H}_i(\vec{z},w)$ and suppress a suitable polynomial bound which can be found easily.

- $F(\vec{z},\epsilon)=G(\vec{z})$
- $F(\vec{z},\mathsf{s}_i w)=\begin{cases}\tilde{H}_i(\vec{z},w), & \text{if } F(\vec{z},w)_0=1 \text{ and } \tilde{H}_i(\vec{z},w)_0\neq 1\\ \langle 1,\langle \mathsf{s}_{\mathsf{bit}(\epsilon,\tilde{H}_i(\vec{z},w)_{1,0})}F(\vec{z},w)_{1,0}, \tilde{H}_i(\vec{z},w)_{1,1}\rangle\rangle, & \text{if } F(\vec{z},w)_0=1 \text{ and } \tilde{H}_i(\vec{z},w)_0=1\\ F(\vec{z},w), & \text{else}\end{cases}$

The function F is in $\mathfrak{W}_3$ because of Ishihara's result delivering the equivalence of $\mathfrak{A}_3$ and $[0,I,\mathsf{S}_0,\mathsf{S}_1,\#,COMP,BRN]$, where BRN denotes bounded recursion on notation. Again, an external induction on the value of w yields the correctness of the realisation function. $\square$

The previous lemma implies together with the lower bound lemma the proof theoretic characterisation of the theories:

Theorem 12

- *The provably total functions of* LogT *are exactly the functions in the logarithmic hierarchy.*
- *The provably total functions of* AlogT *are exactly the functions computable in alternating logarithmic time.*
- *The provably total functions of* PT *are exactly the functions computable in polynomial time.*

3. A new safe function algebra for logspace

We define a two-sorted algebra $\mathfrak{LS}$ of logspace strength. $\mathfrak{LS}$ merits attention because it allows to describe logspace from natural initial functions with only one recursion scheme that does not contain explicit bounds. It differs from the famous Cook-Bellantoni safe algebra for polynomial time only by restricting case distinction, and by allowing an additional initial function len yielding the length of its input.

Definition 13 *The algebra $\mathfrak{LS}$ is the smallest function algebra (on words) which contains the following initial functions and is closed under the following operations:*

Initial functions

- ϵ, s_0, s_1, p_W *with safe input, and* len *with normal input.*
- *Case distinction for safe arguments.*

$$\mathsf{case}(;y_1,y_2,y_3) := \begin{cases} y_2, & \text{if } y_1 = \epsilon \\ y_3, & \text{else} \end{cases}$$

- *Projections $\pi_i^{n,m}$ with both normal and safe inputs.*

Operations

- *Safe composition.*

$$f(\vec{x};\vec{y}) = h(\vec{g}(\vec{x};);\vec{j}(\vec{x};\vec{y}))$$

- *Safe recursion on notation.*

$$\begin{aligned} f(\epsilon,\vec{x};\vec{y}) &:= g(\vec{x};\vec{y}) \\ f(\mathsf{s}_i w,\vec{x};\vec{y}) &:= h_i(w,\vec{x};f(w,\vec{x};\vec{y}),\vec{y}), \end{aligned}$$

Note that Cook and Bellantoni's safe algebra B allows the following stronger case distinction.

$$\mathsf{case}(;y_1,y_2,y_3) := \begin{cases} y_2, & \text{if } mod_2(y_1) = 0 \\ y_3, & \text{else} \end{cases}$$

In addition, one does not need to include len as an initial function of B since it is definable. Apart from these differences, $\mathfrak{LS}$ and B are given in exactly the same way.

Let us mention other safe descriptions of logspace. Bellantoni [3] described logspace as B without s_0. His algebra only delivers the sharply bounded functions computable in logspace.

Another two-sorted algebra was introduced in Oitavem's [19]. Let us compare $\mathfrak{LS}$ to Oitavem's system $Logspace_{CT}$. The main difference is that in $\mathfrak{LS}$, safe recursion is stronger since successor - and predecessor functions can be applied to safe inputs. Therefore, our algebra dispenses with further recursion schemes as Oitavem's log-transition recursion, and safe concatenation recursion on notation. Also some initial functions as multiplication, and iterated predecessor can be dropped. Oitavem's algebra is not contained in $\mathfrak{LS}$: We give an argument for log-transition recursion not being in $\mathfrak{LS}$ in the footnote on page 31.

Møller Neergaard [18] introduced a two-sorted characterisation almost exactly corresponding to B using a composition - and a recursion scheme that allows the use of safe variables only once.

Let us prove now the lower bound for $\mathfrak{LS}$.

Lemma 14 *For each logspace function F there exists an $f \in \mathfrak{LS}$ such that $F(\vec{x}) = f(\vec{x};)$ for all $\vec{x} \in \mathbb{W}$.*

Proof.

Let us introduce the word algebra

$$[\epsilon, I, \mathsf{s}_0, \mathsf{s}_1, \mathsf{len}, \mathsf{bit}, \times, \mathsf{e}, COMP, CRN, SBRN],$$

equivalent to Clote's algebra for logspace [7], where $SBRN$ denotes sharply bounded recursion, given as follows.

$$\begin{aligned} F(\epsilon, \vec{y}) &= G(\vec{y}) \mid |M(\epsilon, \vec{y})| \\ F(\mathsf{s}_0(x), \vec{y}) &= H_0(x, \vec{y}, F(x, \vec{y}))) \mid |M(\mathsf{s}_0(x), \vec{y})| \\ F(\mathsf{s}_1(x), \vec{y}) &= H_1(x, \vec{y}, F(x, \vec{y}))) \mid |M(\mathsf{s}_1(x), \vec{y})|, \end{aligned}$$

where

$$x|y = \begin{cases} x, & \text{if } x \preceq y \\ y, & \text{else} \end{cases}$$

Our algebra clearly contains the functions $\epsilon, I, \mathsf{s}_0, \mathsf{s}_1, \mathsf{len}$. Word multiplication is defined via successor, and concatenation as usual for safe function algebras. The eraser is defined as follows using safe case distinction.

$$\begin{aligned}
\mathsf{e}(\epsilon;\,) &:= \epsilon \\
\mathsf{e}(\mathsf{s}_0 w;\,) &:= \begin{cases} \epsilon, & \text{if } \mathsf{e}(w;\,) = \epsilon \\ \mathsf{s}_0(;\mathsf{e}(w;\,)), & \text{else} \end{cases} \\
\mathsf{e}(\mathsf{s}_1 w;\,) &:= \mathsf{s}_1(;\mathsf{e}(w;\,))
\end{aligned}$$

$\mathfrak{LS}$ is clearly closed under composition. Next, we prove that the algebra is closed under CRN. We use the following auxiliary function b.

$$\begin{aligned}
b(\epsilon; y) &:= \mathsf{s}_0 y \\
b(\mathsf{s}_0 w; y) &:= \mathsf{s}_0 y \\
b(\mathsf{s}_1 w; y) &:= \mathsf{s}_1 y
\end{aligned}$$

Let F be defined by concatenation recursion from G, H_0, H_1. Define f as

$$\begin{aligned}
f(\epsilon, \vec{x};\,) &:= g(\vec{x};\,) \\
f(\mathsf{s}_i w, \vec{x};\,) &:= b(h_i(w, \vec{x};\,); f(w, \vec{x};\,))
\end{aligned}$$

Note that the definitions of b and f are given without using safe case distinction. For bit and $SBRN$, we have to do bootstrapping. The following function $\dot{-}(w; y)$, written as $y \dot{-} w$, is contained in $\mathfrak{LS}$.

$$\begin{aligned}
y \dot{-} \epsilon &:= y \\
y \dot{-} \mathsf{s}_i w &:= \mathsf{p}_\mathsf{W}(; y \dot{-} w)
\end{aligned}$$

Now, we can define the characteristic function of the lexicographic ordering $\preceq$ for two normal arguments w, v. If one of the arguments is larger than the other, we give the suitable output. If both inputs are of equal length, we output $h(w, w, v;\,)$ where h is defined as follows:

$$\begin{aligned}
h(\epsilon, w, v;\,) &:= 1 \\
h(\mathsf{s}_i x, w, v;\,) &:= \begin{cases} 0, \text{ if } b(w \dot{-} x; \epsilon) = 1 \wedge b(v \dot{-} x; \epsilon) = 0 \\ 1, \text{ if } b(w \dot{-} x; \epsilon) = 0 \wedge b(v \dot{-} x; \epsilon) = 1 \\ h(x, w, v;\,), \text{ else} \end{cases}
\end{aligned}$$

The necessary case distinctions are justified using the case distinction implicit in the recursion schema. We define exp as follows.

$$exp(\epsilon, x;) := \epsilon$$
$$exp(\mathsf{s}_i w, x;) := \begin{cases} \mathsf{s}_i w, & \text{if } |\mathsf{s}_i w| \preceq x \\ exp(w, x;), & \text{else} \end{cases}$$

Again, the necessary case distinctions are justified using the case distinction implicit in the recursion schema. The following modified bit function is member of $\mathfrak{LS}$.

$$\mathsf{bit}^*(x, y;) := b(y \dot{-} x; \epsilon)$$

This allows to define the usual bit function.

$$\mathsf{bit}(x, y;) := \mathsf{bit}^*(exp(y, x;), y;)$$

Now, we show that $\mathfrak{LS}$ is closed under sharply bounded recursion. Let F be defined by sharply bounded recursion from G, H_0, H_1 with sharp bound M, where corresponding functions g, h_0, h_1, m are given by induction hypothesis. We show $f \in \mathfrak{LS}$ for a function f such that $F(w, \vec{x}) = |f(w, \vec{x};)|$ which immediately yields the claim. The idea is to determine in each recursion step the length of the recursion argument, which can be assumed to be normal in a certain sense, to apply h_i, and to expand the output using exp.

The function $step_i(w, v, \vec{x}; y)$ we will define below allows to transfer the recursion step of F to f. Its intended first input w is a term bounding the recursion. Its second input corresponds to the recursion argument of f. Its safe argument y is intended to equal $f(v, \vec{x};)$, the other arguments represent side arguments of the recursion. $step_i$ counts down its recursion argument w until we arrive at a w' with $|w'| = |y|$. At this point, the step function h_i representing H_i will be applied with last argument $|w'|$. Finally, $f(\mathsf{s}_i v, \vec{x};)$ is constructed using the function exp with bound $m(\mathsf{s}_i v, \vec{x};)$ as first argument.

Accordingly, $step_i$ is given as follows for $i = 0, 1$.

$$step_i(\epsilon, v, \vec{x}; y) := exp(m(\mathsf{s}_i v, \vec{x};), h_i(v, \vec{x}, \epsilon;);)$$
$$step_i(\mathsf{s}_j w, v, \vec{x}; y) := \begin{cases} step_i(w, v, \vec{x}; y), & \text{if } y \dot{-} w = \epsilon \\ exp(m(\mathsf{s}_i v, \vec{x};), h_i(v, \vec{x}, |\mathsf{s}_j w|;);), & \text{else} \end{cases}$$

In the definition of $step_i$, the use of safe case distinction is essential. The requested function f is defined as follows.

$$f(\epsilon, \vec{x};) := exp(m(\epsilon, \vec{x};), g(\vec{x};);)$$
$$f(\mathsf{s}_i v, \vec{x};) := step_i(m(v, \vec{x};), v, \vec{x}; f(v, \vec{x};))$$

This implies the lower bound for $\mathfrak{LS}$. $\Box$

Let us switch to the upper bound proof which we prove for an extension $\mathfrak{LS}'$ of $\mathfrak{LS}$ formulated in analogy to Bellantoni's BC [3]. As BC, also $\mathfrak{LS}'$ separates not only its input but also its output into safe and normal. $\mathfrak{LS}'$ will be useful for the realisation of theories of logspace strength formulated in the next section. In addition, it can be seen easily that Oitavem's algebra $Logspace_{CT}$ can be embedded into $\mathfrak{LS}'$.

Definition 15 *The algebra $\mathfrak{LS}'$ is the smallest function algebra (on words) which contains the following initial functions and is closed under the following operations:*

Initial functions

- ϵ, s_0, s_1, p_W *with safe input and safe output.*
- len *with normal input and normal output.*
- $\pi_i^{n,m}$ *(projections) with safe output and both normal and safe inputs.*
- $\mathsf{init}(x;y)$ *with normal output which returns the* $\mathsf{len}(\mathsf{len}(x))$ *most significant bits of* y*. Usually, we write* y/x *for* $\mathsf{init}(x;y)$*.*

Operations

- *Composition*
$$f(\vec{x};\vec{y}) = h(\vec{g}(\vec{x};\vec{y});\vec{j}(\vec{x};\vec{y})),$$
where the g_i have normal - and the j_i safe output. f has the same sort of outputs as h.
- *Simultaneous safe recursion on notation defined as follows for $1 \leq j \leq m$, $i = 0, 1$*
$$\begin{aligned} f_j(\epsilon,\vec{x};\vec{y}) &:= g_j(\vec{x};\vec{y}) \\ f_j(\mathsf{s}_i w,\vec{x};\vec{y}) &:= h_{j,i}(w,\vec{x};f_1(w,\vec{x};\vec{y}),\cdots,f_m(w,\vec{x};\vec{y}),\vec{y}), \end{aligned}$$
where $g_1,\cdots,g_m$, $h_{1,0},\cdots,h_{m,0},h_{1,1},\cdots,h_{m,1}$ have safe output. The f_j have safe output.
- *Raising: from $f(\vec{x};)$ with safe output obtain $f^\nu(\vec{x};)$ with normal output.*

The function algebra $\mathfrak{LS}'$ contains $\mathfrak{LS}$ since it can define the function case using the function init, and a case distinction $d(x; y_1, y_2)$ with

$$d(x; y_1, y_2) := \begin{cases} y_1, & \text{if } x = \epsilon \\ y_2, & \text{else ,} \end{cases}$$

which can be defined easily using safe recursion on notation.

Note, that because of projections, functions with normal instead of safe output are admissible in all schemes. $\mathfrak{LS}'$ and BC are formulated very similarly, the most important difference is that in $\mathfrak{LS}'$ only a *sharply* bounded segment of a safe input can by shifted to the normal side, whereas the function *mod* in BC allows to shift a bounded segment. Note that it is important that $\mathfrak{LS}'$ shifts *initial* segments since otherwise an embedding of Cook and Bellantoni's B is clearly possible. The main theorem, stated below, immediately implies that $\mathfrak{LS}'$ is contained in logspace.

Theorem 16 *Let $f(\vec{x}; \vec{y})$ be an element of $\mathfrak{LS}'$ where $\vec{y} := y_1, \cdots, y_n$. Then there exist logspace computable functions $ch, del, const$ on words, and unary (monotone) polynomials Q, M on words such that for all $\vec{x}, \vec{y} \in \mathbb{W}$ the following properties hold. We write $M(\vec{x})$ for $M(max(\vec{x}))$, and $\vec{y}/M(\vec{x})$ for $y_1/M(\vec{x}), \cdots, y_n/M(\vec{x})$. The output of ch is displayed as a number.*

- $0 \leq ch(\vec{x}, \vec{y}/M(\vec{x}), |\vec{y}|) \leq n$.
- *If f has normal output, $ch(\vec{x}, \vec{y}/M(\vec{x}), |\vec{y}|) = 0$.*
- *If $ch(\vec{x}, \vec{y}/M(\vec{x}), |\vec{y}|) = 0$, we have*

$$f(\vec{x}; \vec{y}) = const(\vec{x}, \vec{y}/M(\vec{x}), |\vec{y}|).$$

- *If $0 < ch(\vec{x}, \vec{y}/M(\vec{x}), |\vec{y}|) = k \leq n$, we have*

$$f(\vec{x}; \vec{y}) = \big(y_k \dot{-} del(\vec{x}, \vec{y}/M(\vec{x}), |\vec{y}|)\big) * const(\vec{x}, \vec{y}/M(\vec{x}), |\vec{y}|).$$

- *$del(\vec{x}, \cdots), const(\vec{x}, \cdots) \leq Q(\vec{x})$, where $\cdots$ stands for an arbitrary input of fitting size.*

Let us explain now the functions $ch, del, const$ mentioned in the theorem. First, consider the case where $1 \leq ch(\vec{x}, \vec{y}/M(\vec{x}), |\vec{y}|) = i \leq n$. Then, $f(\vec{x}; \vec{y})$ is calculated by first deleting $|del(\vec{x}, \vec{y}/M(\vec{x}), |\vec{y}|)|$ bits from y_i. (In the whole proof only the lengths of the *del*-outputs matter.) Then, we concatenate

$const(\vec{x}, \vec{y}/M(\vec{x}), |\vec{y}|)$ (*const* stands for construct). If the value of the choice function is zero, $f(\vec{x}; \vec{y})$ is given as $const(\vec{x}, \vec{y}/M(\vec{x}), |\vec{y}|)$.

Note that in both cases, $f(\vec{x}; \vec{y})$ fully depends only on a single safe input. For the other safe inputs, only their length, and sharply bounded initial segments matter. Also the chosen safe input can only be manipulated in a very restricted way. This will allow us to simulate safe recursion essentially by sharply bounded recursion since only a very small amount of information about the intermediate values has to be stored.

In the theorem, we additionally claim

$$del(\vec{x}, \cdots), const(\vec{x}, \cdots) \leq Q(\vec{x}).$$

This property is important to prove that the functions in $\mathfrak{LG}'$ have at most polynomial growth. To treat recursion, it is crucial that the polynomials M and Q do not depend on the safe arguments $\vec{y}$. Let us start with the proof of the theorem.

Proof. Let us first introduce some notations used in this section. We work with the analogues on words of the arithmetical plus and minus operations, given as $+, -$ using again the isomorphism between $(\mathbb{W}, \preceq)$ and $(\mathbb{N}, \leq)$. As for numbers, these operations can be extended within the logarithmic hierarchy to negative words, displayed as $-w$, using a natural coding. Also the ordering $\preceq$ is extended as expected to negative words. We work for technical reasons with a bit function on words, which enumerates the bits in the opposite way as before, which we call bit as well. The most significant bit of w is given as $\mathsf{bit}(0, w)$ and the least significant as $\mathsf{bit}(|w|, w)$.

We prove the claim by induction on the complexity of f, and detail the most interesting steps.

For initial functions, the claim clearly holds. E.g. for $\mathsf{init}(x; y)$, which has normal output, *const* is given as y/x.

Composition

Assume that f is defined by composition as follows for $\vec{j}$ containing m components.

$$f(\vec{x}; \vec{y}) = g(\vec{h}(\vec{x}; \vec{y}); \vec{j}(\vec{x}; \vec{y}))$$

Let us first reflect the induction hypothesis for the components of $\vec{j}$. $j_i(\vec{x}, \vec{y})$ for $1 \leq i \leq m$ is given as a sum of at most three summands (always relative to $\dot{-}$ and $*$). If $0 < ch(\vec{x}, \vec{y}/M(\vec{x}), |\vec{y}|) = k \leq n$, this sum is composed of y_k, del_{j_i}, and $const_{j_i}$ (with suppressed inputs). The same argument for the function g implies that $f(\vec{x}; \vec{y})$ is given as a sum of at most 5 summands

containing at most two *del* - and two *const* summands. The strategy is first to produce these summands as logspace functions of our initial inputs. Then, we combine them to get del_f and $const_f$.

In the following, we define auxiliary functions for arbitrary inputs $\vec{b}, \vec{c}, \vec{d}$. We motivate them for the case that the inputs $\vec{b}, \vec{c}, \vec{d}$ are given as intended, i.e. as $\vec{x}, \vec{y}/M_f(\vec{x}), |\vec{y}|$, where $M_f(x)$ is given as $M_g(Q_h(x))$, where Q_h denotes a bounding polynomial for the bounds Q_{h_i}. The following term equals $h_i(\vec{x}; \vec{y})$ for intended inputs according to the induction hypothesis.

$$const_{h_i}(\vec{b}, \vec{c}/M_{h_i}(\vec{b}), \vec{d}) := q_i$$

Next, we have to collect information about the $j_i(\vec{x}; \vec{y})$. We abbreviate

$$ch_{j_i}(\vec{b}, \vec{c}/M_{j_i}(\vec{b}), \vec{d})$$

as k_i. We consider the case $1 \leq k_i \leq n$, and define ℓ_i as follows.

$$\ell_i := max(\epsilon, d_{k_i} - |del_{j_i}(\vec{b}, \vec{c}/M_{j_i}(\vec{b}), \vec{d})|),$$

where max denotes a maximum function relying on the lexicographic order extended to negative words, as explained at the beginning of the proof. If $1 \leq k_i \leq n$ does not hold, we define ℓ_i as ϵ. The length of $j_i(\vec{x}; \vec{y})$ is given as

$$\tilde{d}_i := \ell_i + |const_{j_i}(\vec{b}, \vec{c}/M_{j_i}(\vec{b}), \vec{d})|.$$

We can now define a function F_i which constructs initial segments of $j_i(\vec{x}; \vec{y})$ as follows.

$$F_i(\epsilon, \vec{b}, \vec{c}, \vec{d}) := \epsilon$$

$$F_i(\mathsf{s}_i z, \vec{b}, \vec{c}, \vec{d}) := \begin{cases} \mathsf{s}_{\mathsf{bit}(|\mathsf{s}_i z|, c_{k_i})} F_i(z, \vec{b}, \vec{c}, \vec{d}), & \text{if } |\mathsf{s}_i z| \preceq \ell_i \\ \mathsf{s}_{\mathsf{bit}(|\mathsf{s}_i z| - \ell_i, const_{j_i}(\vec{b}, \vec{c}/M_{j_i}(\vec{b}), \vec{d})} F_i(z, \vec{b}, \vec{c}, \vec{d}), & \text{if } \ell_i \prec |\mathsf{s}_i z| \preceq \tilde{d}_i \\ F_i(z, \vec{b}, \vec{c}, \vec{d}), & \text{else} \end{cases}$$

Note that an initial segment of $j_i(\vec{x}; \vec{y})$ is constructed by first reading bits of the chosen safe input, and then reading the word provided by $const_{j_i}$. For $1 \leq i \leq m$, $F_i(|M_g(\vec{q})|, \vec{b}, \vec{c}, \vec{d})$ produces initial segments of sufficient length for the later application of g (the q_i are defined above). Now, we can calculate the safe input the function g chooses as follows.

$$ch_g(\vec{q}, \vec{F}(|M_g(\vec{q})|, \vec{b}, \vec{c}, \vec{d}), \vec{\tilde{d}}) := ch_g$$

We are ready to define the searched functions ch_f, del_f, $const_f$. This will be done using a case distinction on the value of ch_g.

First, we assume $1 \leq ch_g \leq m$. Then, $ch_f(\vec{b}, \vec{c}, \vec{d})$ is given as

$$ch_{(j_{ch_g})}(\vec{b}, \vec{c}/M_{j_{ch_g}}(\vec{b}), \vec{d}) := k.$$

This means that the safe input we use as first summand when writing $f(\vec{x}; \vec{y})$ as sum of five summands is the safe input the chosen function j_{ch_g} choses for $\vec{b}, \vec{c}, \vec{d}$ given as intended. $del_f(\vec{b}, \vec{c}, \vec{d})$ is defined as

$$\begin{aligned}
&del_{j_k}(\vec{b}, \vec{c}/M_{j_k}(\vec{b}), \vec{d}) * \\
&(del_g(\vec{q}, \vec{F}(|M_g(\vec{q})|, \vec{b}, \vec{c}, \vec{d}), \vec{\vec{d}}) \dot{-} \\
&const_{j_k}(\vec{b}, \vec{c}/M_{j_k}(\vec{b}), \vec{d})),
\end{aligned}$$

$const_f(\vec{b}, \vec{c}, \vec{d})$ as

$$\begin{aligned}
&\big(const_{j_k}(\vec{b}, \vec{c}/M_{j_k}(\vec{b}), \vec{d}) \dot{-} \\
&del_g(\vec{q}, \vec{F}(|M_g(\vec{q})|, \vec{b}, \vec{c}, \vec{d}), \vec{\vec{d}})\big) * \\
&const_g(\vec{q}, \vec{F}(|M_g(\vec{q})|, \vec{b}, \vec{c}, \vec{d}), \vec{\vec{d}})
\end{aligned}$$

Assume now that $1 \leq ch_g \leq m$ does not hold. Then, $ch_f(\vec{b}, \vec{c}, \vec{d})$ is given as 0. $del_f(\vec{b}, \vec{c}, \vec{d})$ is given as ϵ, and $const_f(\vec{b}, \vec{c}, \vec{d})$ as

$$const_g(\vec{q}, \vec{F}(|M_g(\vec{q})|, \vec{b}, \vec{c}, \vec{d}), \vec{\vec{d}})$$

Finally, we define the function Q_f as $Q_j(\vec{b}) * Q_g(Q_h(\vec{b}))$ (independent of ch_g), where Q_h, Q_j denote bounding polynomials for the Q_{h_i}, Q_{j_i}, respectively.

Recursion

First, we analyse the case of ordinary safe recursion. Then, it is easy to generalise the argument to simultaneous safe recursion. Assume that f is defined as follows.

$$\begin{aligned}
f(\epsilon, \vec{x}; \vec{y}) &:= g(\vec{x}; \vec{y}) \\
f(\mathsf{s}_i z, \vec{x}; \vec{y}) &:= h_i(z, \vec{x}; f(z, \vec{x}; \vec{y}), \vec{y})
\end{aligned}$$

Let us give the intuitive reason why this recursion goes through in logspace. Because our induction hypothesis about g, h_0, h_1, we know that for any $z \in \mathbb{W}$

we can write $f(z, \vec{x}; \vec{y})$ ultimately as sum of at most one safe input and several *del* and *const* terms, depending on the order of applications of h_0, h_1 necessary to calculate $f(z, \vec{x}; \vec{y})$. These terms only depend on the length, and a in z and $\vec{x}$ sharply bounded initial segment of the intermediate values $f(z', \vec{x}; \vec{y})$ with $z' \subseteq z$, which allows to simulate safe recursion using sharply bounded recursion.

Let us give the calculations in more detail. First, we have to make sure until which bit we have to know the safe inputs $\vec{y}$ to compute all necessary $ch, del, const$ terms during the recursion. It is easy to see that a bounding polynomial M_f of M_g, M_{h_0}, and M_{h_1} is sufficient.

We define an auxiliary function $H(w, a, \vec{b}, \vec{c}, \vec{d})$ such that

$$H(w, z, \vec{x}, \vec{y}/M_f(w, \vec{x}), |\vec{y}|) := H$$

contains information about $f(z, \vec{x}; \vec{y})$ for $z \subseteq w$. We sketch the definition of H, and explain the meaning of its output for the intended input, then we give a precise definition of H. The output of H will always be a tuple of five words. The first component is displayed as a number, and tells us for $z = \mathsf{s}_i z' \subseteq w$ which safe argument of $h_i(z', \vec{x}; f(z', \vec{x}; \vec{y}), \vec{y})$ is used when writing $f(z, \vec{x}; \vec{y})$ as sum of three summands using the *del-const* unfolding. For $z = \epsilon$, we use g instead of h_i. If there is no such safe input, we stipulate $H_0 = 0$.

H_1 is displayed as number as well, and tells us which safe input y_ℓ of $\vec{y}$ is needed when we write $f(z, \vec{x}; \vec{y})$ as a totally unfolded *del-const* sum. This means, for $z = \mathsf{s}_i z' \subseteq w$ we write $f(z, \vec{x}; \vec{y})$ first (if possible) as $f(z', \vec{x}; \vec{y})$ minus a *del*, plus an *const* summand, then unfold $f(z', \vec{x}; \vec{y})$ analogously, and so on. If there is no such safe input, we stipulate $H_1 = 0$. For $z = \epsilon$, $H_1 = H_0$.

We assume $z = \mathsf{s}_i z'$ and abbreviate

$$|del_{h_i}(z', \vec{x}, f(z', \vec{x}; \vec{y})/M_{h_i}(z', \vec{x}), \vec{y}/M_{h_i}(z', \vec{x}), |f(z', \vec{x}; \vec{y})|, |\vec{y}|)|$$

as *del*. H_2 is the length of $(f(z', \vec{x}; \vec{y}) \dot{-} del)$ minus the length of y_ℓ, defined above. Note that H_2 is possibly a negative word. If no y_ℓ exists H_2 is ϵ. For $z = \epsilon$ H_2 is given analogously but using del_g, and y_ℓ instead of $f(z', \vec{x}; \vec{y})$. The fourth component H_3 gives the length of $f(z, \vec{x}; \vec{y})$ minus the length of y_ℓ, or simply the length of $f(z, \vec{x}; \vec{y})$ if y_ℓ does not exist. The reason for giving H_2 and H_3 as lengths relative to the length of y_ℓ is to insure that they can be sharply bounded by a term only depending on the normal arguments of f.

H_4 contains the $||M_f(w, \vec{x})||$ most significant bits of $f(z, \vec{x}; \vec{y})$.

In the following, we give a precise definition of H and argue that it is logspace computable. We define $H(w, a, \vec{b}, \vec{c}, \vec{d})$ by sharply bounded induction on a. We suppress a bound in the following and argue in the end that it can be found easily. We let n denote the number of components of $\vec{c}$ and of $\vec{d}$.

The first and second component of $H(w, \epsilon, \vec{b}, \vec{c}, \vec{d})$ are defined as

$$ch_g(\vec{b}, \vec{c}/M_g(\vec{b}), \vec{d}) := k|n$$

for $|$ being the cut function defined on page 20. We suppress such bounds for the first and second component in the following. For $1 \leq k \leq n$ the third component is given by

$$max(-d_k, -|del_g(\vec{b}, \vec{c}/M_g(\vec{b}), \vec{d})|) := q_2.$$

We abbreviate $const_g(\vec{b}, \vec{c}/M_g(\vec{b}), \vec{d})$ as $const$. The fourth component is given as $q_2 + |const|$.

The fifth component is constructed by glueing c_k and $const$ together, very similarly as on page 26. If $1 \leq k \leq n$ does not hold, we output ϵ, $|const|$, $const/M_f(w, \vec{b})$ as third until fifth component.

Now, we show how to calculate $H(w, \mathsf{s}_i a, \vec{b}, \vec{c}, \vec{d}) := r$ from

$$w, a, \vec{b}, \vec{c}, \vec{d}, H(w, a, \vec{b}, \vec{c}, \vec{d}) := q$$

in logspace. For convenience, in the following for all words i if $1 \leq i \leq n$ does not hold, we let d_i denote ϵ. r_0 is given as

$$ch_{h_i}(a, \vec{b}, q_4/M_{h_i}(a, \vec{b}), \vec{c}/M_{h_i}(a, \vec{b}), d_{q_1} + q_3, \vec{d}) := k.$$

Note that we have inserted the values $q_4/M_{h_i}(a, \vec{b})$ and $d_{q_1} + q_3$ corresponding to a sharply bounded initial segment, and the length of an intermediate value of the recursion for intended inputs. r_1 is given as q_1 if r_0 equals 1, as $r_0 - 1$ if $r_0 > 1$, and as 0 else. For the other components, we use the following case distinction: We first assume that k equals 1. Then, r_2 is given as

$$max(-d_{q_1}, q_3 - |del_{h_i}(a, \vec{b}, q_4/M_{h_i}(a, \vec{b}), \vec{c}/M_{h_i}(a, \vec{b}), d_{q_1} + q_3, \vec{d})|),$$

r_3 is given as

$$r_2 + |const_{h_i}(a, \vec{b}, q_4/M_{h_i}(a, \vec{b}), \vec{c}/M_{h_i}(a, \vec{b}), d_{q_1} + q_3, \vec{d})|.$$

We abbreviate the $const$-term above as $const$. r_4 is defined by glueing together q_4 and $const$, similarly as on page 26.

If we have $2 \leq k \leq n+1$, the components r_2, r_3, r_4 are given in a very similar way as for $H(w, \epsilon, \vec{b}, \vec{c}, \vec{d})$ since in this case we build a new *del-const* sum from a safe input. *In all other cases*, we output ϵ, $|const|$, $const/M_f(w, \vec{b})$ as r_2, r_3, r_4.

Let us finally say a word about the sharp bound we have to deliver for this recursion. It suffices to show that all components of $H(w, a, \vec{b}, \vec{c}, \vec{d})$ are sharply bounded by polynomials of $w, a, \vec{b}$ since we are using a linear pairing function. For the first, second, and fifth component this is clear. For the third and fourth component, we use a bounding polynomial

$$\left[Q_g(\vec{b}) * (Q_h(a, \vec{b}) \times a)\right] \times c,$$

where the constant c depends on the exact definition of the coding for negative words. This concludes the proof that the auxiliary function H is logspace computable.

It is easy to define ch_f, del_f, $const_f$ from H: $ch_f(a, \vec{b}, \vec{c}, \vec{d})$ is simply given as $H(a, a, \vec{b}, \vec{c}, \vec{d})_1 := k$. del_f and $const_f$ are constructed using a case distinction on k. First, we assume $1 \leq k \leq n$. Let us find the smallest $z \subseteq a$ such that $z \subset z' \subseteq a$ implies $H(a, z', \vec{b}, \vec{c}, \vec{d})_0 = 1$ [5]. For $z \subseteq z' \subseteq a$, $f(z', \vec{x}; \vec{y})$ is given as a *del-const* sum starting at y_k. Find the minimal

$$H(a, z', \vec{b}, \vec{c}, \vec{d})_2$$

for $z \subseteq z' \subseteq w$ which we abbreviate as q. $del_f(a, \vec{b}, \vec{c}, \vec{d})$ is given as

$$exp(Q_f(w, \vec{x}), max(\epsilon, -q)),$$

where exp is defined on page 21.

Next, we show how $const_f(a, \vec{b}, \vec{c}, \vec{d})$ can be calculated in logspace, by calculating its i-th bit in logspace for an arbitrary i. We abbreviate $|y_k \dot{-} del_f(a, \vec{b}, \vec{c}, \vec{d})|$ as ℓ. We search the largest $z \subseteq z' \subseteq w$ such that

$$d_k + H(a, z', \vec{b}, \vec{c}, \vec{d})_2 \prec \ell + i \preceq d_k + H(a, z', \vec{b}, \vec{c}, \vec{d})_3.$$

If no such z' exists, i exceeds the length of $const_f(a, \vec{b}, \vec{c}, \vec{d})$. Assume $z' = \mathsf{s}_i v$. Then, we easily find the value of the searched i-th bit by calculating

$$const_{h_i}(v, \vec{b}, r_4/M_{h_i}(v, \vec{x}), \vec{y}/M_{h_i}(v, \vec{x}), d_k + r_3, \vec{d}),$$

[5] Sharply bounded quantification which immediately yields subword quantification is admissible within the logarithmic hierarchy, see Clote's [7].

where $r := H(a, v, \vec{b}, \vec{c}, \vec{d})$. If $z' = \epsilon$, we use $const_g$ instead.

If $1 \leq k \leq n$ does not hold, $const_f$ is defined similarly, and del_f as ϵ. This finishes our argument for the ordinary safe recursion.

Simultaneous safe recursion

We sketch how to produce the functions ch_{f_ℓ}, del_{f_ℓ}, $const_{f_\ell}, M_{f_\ell}, Q_{f_\ell}$ for $1 \leq \ell \leq m$ if f_ℓ is defined by simultaneous safe recursion.

- For all $1 \leq \ell \leq m$, we give the same M_{f_ℓ}, Q_{f_ℓ} which we call M_f, Q_f. We define them as for ordinary safe recursion using bounds that work for all base - and recursion step functions.

- We use again an auxiliary function H'. It contains the same information as H but simultaneously for all f_ℓ with $1 \leq \ell \leq m$. So, e.g., we collect the first $||M_f(w, \vec{x})||$ bits for all f_ℓ. The components of H' can be calculated very similarly as the ones of H.

- ch_{f_ℓ} is given analogously as before. For del_{f_ℓ}, and $const_{f_\ell}$ the complication is that it is not sufficient to know which y_k occurs as first summand in the total unfolding of $f_\ell(z, \vec{x}; \vec{y})$ as sum of *del* and *const* terms. This is so because it does not give us the information in which order which base - and recursion step functions were applied. The problem is solved by tracing back which safe input is needed to write $f_\ell(z, \vec{x}; \vec{y})$ as a *del-const* sum using $H(z, a', \vec{b}, \vec{c}, \vec{d})_0$ for $a' \subseteq z$. Then, following this trace, del_{f_ℓ} and $const_{f_\ell}$ are defined very similarly as before. The back-tracing is possible using constantly bounded recursion.

This concludes the proof of the theorem. □

The lower bound result implies together with the previous theorem the following corollary [6].

Corollary 17 *The normal segments of the algebras $\mathfrak{LS}$ and $\mathfrak{LS}'$ describe exactly the logspace computable functions.*

[6] The technique used to show that $\mathfrak{LS}'$ has strength logspace can also be directly applied to $\mathfrak{LS}$. The dependence of *ch*, *del*, and *const* on sharply bounded initial segments of safe inputs can then be dropped. This immediately implies that Oitavem's log-transition recursion cannot be defined in $\mathfrak{LS}$. Nevertheless, note that it can be defined easily in $\mathfrak{LS}'$. This immediately implies that Oitavem's algebra $Logspace_{CT}$ is contained in $\mathfrak{LS}'$.

4. Two systems of strength logspace

We formalise the algebra $\mathfrak{LS}$ and Clotes's function algebra for logspace [7] with concatenation and sharply bounded recursion within an applicative setting. The theories again contain a predicate for normal - and a predicate for safe words, interpreted similarly as before. In contrast to the theories presented earlier, for logspace strength we allow ordinary one-variable induction schemes.

4.1. Formalising $\mathfrak{LS}$

We introduce the theory LogST, formalising $\mathfrak{LS}$, and prove its lower bound in terms of provably total functions. We deliver the upper bound proof for a stronger system LogST' formalising $\mathfrak{LS}'$.

Definition 18 *The theory* LogST *is the theory* LogT *with the following modifications.*

- *The induction axiom is replaced by* V*-induction, defined as follows for* $x \notin FV(r)$*:*

$$r\epsilon \in \mathsf{V} \wedge (\forall x \in \mathsf{W})(rx \in \mathsf{V} \to r(\mathsf{s}_i x) \in \mathsf{V}) \to (\forall x \in \mathsf{W})(rx \in \mathsf{V})$$

- *We drop the axioms for bit, concatenation, multiplication, and eraser.*
- *Let* case *be a closed term corresponding to the function* $\mathsf{case} \in \mathfrak{LS}$ *for which the usual elementary properties are provable in* LogT*. Then we have as additional axioms the following.*
  - $x, y, z \in \mathsf{V} \to \mathsf{case}(x, y, z) \in \mathsf{V}$
  - $x, y \in \mathsf{V} \to \mathsf{case}(x, y, \epsilon) = x$
  - $x, y, z \in \mathsf{V} \to (\mathsf{case}(x, y, z) = y \vee z = \epsilon)$

The following lemma is proved by straightforward induction on the complexity of F.

Lemma 19 *For any* $F(\vec{x}; \vec{y}) \in \mathfrak{LS}$ *there is an* L *term* t_F *with*

- $\mathsf{LogST} \vdash \vec{x} \in \mathsf{W} \wedge \vec{y} \in \mathsf{V} \to t_F(\vec{x}, \vec{y}) \in \mathsf{V}$
- $\mathcal{M}(\lambda\eta) \vDash t_F(\vec{\overline{w}}, \vec{\overline{v}}) = \overline{F(\vec{w}; \vec{v})}$*, where* $\mathcal{M}(\lambda\eta)$ *denotes the standard open term model.*

Now, we switch to the theory LogST' formalising $\mathfrak{LS}'$ which is formulated with a more flexible induction scheme.

Definition 20 *The theory* LogST' *is the theory* LogT *with the following modifications.*

- *The induction axiom is replaced by positive* W*-free induction.*
- *We drop the axioms for bit, concatenation, multiplication, and eraser.*
- *Let* init *be a closed term corresponding to the function* $\mathsf{init} \in \mathfrak{LS}'$ *for which the usual elementary properties are provable in* LogT*. Then we have as additional axiom the following.*

$$x \in \mathsf{W} \wedge y \in \mathsf{V} \to \mathsf{init}(x, y) \in \mathsf{W}$$

A term case' corresponding to case in LogST can be defined in LogST' as follows.

$$\mathsf{case}' := \lambda x.\lambda y.\lambda z.\mathsf{d}_{\mathsf{W}}(x, y, \epsilon, \mathsf{init}(1, z))$$

The upper bound proof for LogST' is technically involved. The main problem is, that a pairing function p for safe inputs is not available. This is a consequence of theorem 16 as we will argue in the following. Assume that p is such a pairing function. Independently of whether we can write $\mathsf{p}(; y_1, y_2)$ according to the theorem as *const* term, or as sum of three summands, it can be bounded by one of its safe inputs concatenated with a fixed polynomial in the length of both safe inputs. This condition cannot be fulfilled for arbitrary y_1, y_2.

Let us now point out, why it is a problem not to have a pairing function for safe inputs: Complex formulas without occurrences of W have to be realised by a safe input of the realisation function to allow induction. Nevertheless, without pairing - and projection functions, we are unable to access the realisers of the components of the formula which makes the realisation approach impossible.

Cantini presented in [6] a realisation approach that also handles the problem of the missing pairing function for safe inputs. Nevertheless, we present an alternative realisation approach, which deals with all V-atoms separately, and uses a set of realisation functions such that each yields only the realiser of a single V-atom. In contrast to Cantini's approach, vector-valued functions are not necessary, and also proofs containing formulas with both, W and V-occurrences, can be treated. This allows the realisation of a more general V-elimination rule in comparison to Cantini's [6]. The presented approach also solves a technical problem Cantini's approach faces for the realisation of disjunctions whose realisers may have different types.

Let us give the details of the realisation approach. We number the V-atoms of a formula D from the left to the right and call the k-th such atom D_k. If $\rho \ \mathfrak{r} \ D$ or $\rho \ \mathfrak{R} \ D$ holds, for the realisation relations $\mathfrak{r}$ and $\mathfrak{R}$ defined on page 13, we can speak of the set of realised atoms as on page 14. We write $\rho \ \mathfrak{r}^\star \ D_k$ or $\rho \ \mathfrak{R}^\star \ D_k$, respectively, if D_k belongs to the set of realised atoms. If $\rho \ \mathfrak{R} \ D$ holds, we also use the expression "$\rho \ \mathfrak{R}^\star \ D_k$ by w" for a word w if w is the component of ρ realising the atom D_k.

We define a realisation relation $\mathcal{S}$ for sequents which realises the V-atoms separately. We let $< \cdot, \cdot >$ denote a standard set theoretic pairing function.

Definition 21 *Let Γ be given as $A_1, \cdots, A_n$ (with V-atoms given as $A_{i,1}, \cdots, A_{i,k_i}$). $\rho \ \mathcal{S} \ \Gamma$ holds exactly if*

- *ρ is of the form*

$$<< v_1, \cdots, v_n >, < w_{1,1}, \cdots, w_{1,k_1}, \cdots, w_{n,k_n} >>$$

  where k_i equals the number of V-atoms of A_i for each $1 \leq i \leq n$. (k_i might equal zero.)

- $< v_1, \cdots, v_n > \ \mathfrak{r} \ \Gamma$.

- *For any $1 \leq i \leq n$ there is a (unique) $\wp$ with $v_i = \wp^{A_i}$ and $\wp \ \mathfrak{R} \ A_i$ such that*

$$(v_i \ \mathfrak{r}^\star \ A_{i,j} \Leftrightarrow) \wp \ \mathfrak{R}^\star \ A_{i,j} \Leftrightarrow \wp \ \mathfrak{R}^\star \ A_{i,j} \text{ by } w_{i,j}.$$

Note that the use of two realisation relations $\mathfrak{r}$ and $\mathfrak{R}$ and their interplay using the projection function was already used in section 2 for the realisation of the weaker theories. Here, the new idea is the individual realisation of V-atoms.

Note that for formulas A_i without occurrence of V the third property is fulfilled for $\wp = v_i$ since the biconditionals hold trivially in this case as no $A_{i,j}$ exist. For the realisation relation $\mathcal{S}$ the usual properties with respect to quantification and equality of terms hold as a consequence of these properties for $\mathfrak{R}$, $\mathfrak{r}$.

Lemma 22 *For the realisation relation $\mathcal{S}$ the following properties hold. We use $\vec{s} = \vec{t}$ as an abbreviation of $s_0 = t_0 \wedge \cdots \wedge s_n = t_n$.*

- $\rho \ \mathcal{S} \ (\exists x) A[x] \Leftrightarrow \rho \ \mathcal{S} \ A[t]$ *for some term t*

- $\rho \ \mathcal{S} \ (\forall x) A[x] \Leftrightarrow \rho \ \mathcal{S} \ A[u]$

- $\rho\ \mathcal{S}\ A[\vec{s}] \Leftrightarrow \rho\ \mathcal{S}\ A[\vec{t}]$

Proof. Assume $\rho\ \mathcal{S}\ (\exists x)A[x]$. The realisation property 3 implies the existence of a $\wp$ such that $\wp\ \mathfrak{R}\ (\exists x)A[x]$ and $\wp^{(\exists x)A[x]} = \rho_{0,0}$. This implies $\wp\ \mathfrak{R}\ A[t]$ for some term t. The properties of the projection function imply $\wp^{A[t]} = \rho_{0,0}$ and $\rho_{0,0}\ \mathfrak{r}\ A[t]$. Therefore, realisation properties 2 and 3 hold relative to t. Also property 1 clearly holds. This implies the direction from left to right for the first claim. The other direction and the other claims clearly hold because of the analogous properties for $\mathfrak{R}$, $\mathfrak{r}$ and the properties of the projection function. □

Note that the lemma above also holds with side formulas present.

We are ready to state the main theorem for a total sequent-style formalisation of LogST, called LogST as well, which is constructed as for the theories presented in section 2. An $\mathcal{S}$ -realiser $\rho :=< \rho_0, \rho_1 >$ is inserted into realisation functions of $\mathfrak{LS}$ as

$$(\rho_{0,0}, \rho_{0,1}, \cdots ; \rho_{1,0}, \rho_{1,1}, \cdots)$$

which we abbreviate as $(\rho_0; \rho_1)$.

Theorem 23 *Assume that* Γ *and* $\Delta \equiv D_1, \cdots, D_m$ *(with* V*-atoms given as* $D_{i,1}, \cdots, D_{i,k_i}$*) are positive sequences of formulas. Assume* LogST $\vdash \Gamma \Rightarrow \Delta$ *with a proof containing only positive formulas. Then there are* $\mathfrak{LS}$*-functions* F *with normal output and* $f_{<1,1>}, \cdots, f_{<1,k_1>}, \cdots, f_{<m,k_m>}$ *with safe outputs such that for all substitutions* $[\vec{s}]$ *and for all* $\rho\ \mathcal{S}\ \Gamma[\vec{s}]$ *the following properties hold.*

- $1 \leq F(\rho_0; \rho_1)_0 := i \leq m$
- $<< F(\rho_0; \rho_1)_1 >, < f_{i,1}(\rho_0; \rho_1), \cdots, f_{i,k_i}(\rho_0; \rho_1) >>\ \mathcal{S}\ D_i$

Proof. We use an induction on the length of the positive proof. The axioms are realised easily.

Let us realise V-elimination given as follows where Γ, Δ does not contain V, and B_{W} is B with all V replaced by W.

$$\frac{\Gamma \Rightarrow B, \Delta}{\Gamma \Rightarrow B_{\mathsf{W}}, \Delta}$$

We assume realisation functions $G, g_{1,1}, \cdots$ for the premise. Let us define F by the following algorithm for a given input $(\rho_0;)$. Let $G(\rho_0;)_0 = 1$. For each $1 \leq i \leq k_1$, replace the ϵ in $G(\rho_0;)_1$ responsible for the i-th V-atom by $g_{1,i}(\rho_0;)$ if there is such an ϵ. (The relevant ϵ's can be found by only

considering the structure of $G(\rho_0;)_1$.) The resulting word α is clearly an $\mathfrak{r}$ realiser of B_{W}. Therefore, we define $F(\rho_0;)$ as $\langle 1, \alpha\rangle$ in this case. If $G(\rho_0;)_0 \neq 1$ we define $F(\rho_0;)$ just as $G(\rho_0;)$.

Let us argue that the function F is in $\mathfrak{LS}$ which follows from α being producible from ρ_0 within $\mathfrak{LS}$: $G(\rho_0;)_1$ delivers normal output, and the $g_{1,i}(\rho_0;)$ can be assumed to deliver normal output as well because of the raising rule. Therefore, the lower bound lemma for $\mathfrak{LS}$ implies that their outputs can be freely used as input for logarithmic space functions. For each $1 \leq i \leq k_1$, we can find the ϵ responsible for the corresponding atom, if it exists, within the logarithmic hierarchy, since this only involves keeping track of the structure of $G(\rho_0;)_1$ relative to the pairing function. Once the positions of the ϵ's are found, the replacements can be clearly executed within the logarithmic hierarchy. Finally, the $f_{i,j}$ are found easily [7].

Induction is realised using the scheme of simultaneous recursion. Assume that the rule has the following form, where we have premise realisation functions $G, g_{1,1}, \cdots, H_i, h_{i,1,1}, \cdots$.

$$\frac{\Gamma \Rightarrow A[0], \Delta \qquad \Gamma, x \in \mathsf{W}, A[x] \Rightarrow A[\mathsf{s}_i x], \Delta}{\Gamma, t \in \mathsf{W} \Rightarrow A[t], \Delta}$$

Remember, that we use the standard linear pairing operation $\langle , \rangle$ which is in the logarithmic hierarchy. To determine, whether the first element of a certain pair is 1, we have to know only finitely many initial bits. Therefore, case distinctions with this property are permitted for safe inputs.

By lemma 11, we find a word c such that for all substitutions $[\vec{s}]$

$$\rho \ \mathfrak{r} \ A[\vec{s}] \Rightarrow \langle 1, \rho\rangle \leq |c|.$$

We define an auxiliary function F', motivated as follows. We keep its output small such that we can transform it into a normal output as demanded for F. This works without a problem as long as main formulas is realised. If in turn a side formula is realised, the produced realiser does not have to be sharply bounded. Therefore, in this case, F' only stores the information in which induction step for the first time a side formula is realised. Now, we define F', and the functions $f_{i,j}$ by simultaneous safe recursion. We abbreviate

$$H_i(\vec{x}, w, \mathsf{init}(c, F'(\vec{x}, w; \vec{y}))_1; \vec{y}, \vec{f_1}(\vec{x}, w; \vec{y}))$$

[7] Note that the realisation of V-elimination is not entirely trivial, because of the asymmetric treatment of the W and V-atoms in our approach. It would be possible to modify it such that single W-atoms are realised by functions with normal instead of safe outputs which would yield a trivial realisation of V-elimination.

as $Q_i(\vec{x}, w; \vec{y})$.

$$F'(\vec{x}, \epsilon; \vec{y}) := \begin{cases} G(\vec{x}; \vec{y}), & \text{if } G(\vec{x}; \vec{y})_0 = 1 \\ \langle 2, \epsilon \rangle, & \text{else} \end{cases}$$

$$F'(\vec{x}, \mathsf{s}_i w; \vec{y}) := \begin{cases} Q_i(\vec{x}, w; \vec{y}), & \text{if } F'(\vec{x}, w; \vec{y})_0 = 1 \wedge Q_i(\vec{x}, w; \vec{y})_0 = 1 \\ \langle 3, |w| \rangle, & \text{if } F'(\vec{x}, w; \vec{y})_0 = 1 \wedge Q_i(\vec{x}, w; \vec{y})_0 \neq 1 \\ F'(\vec{x}, w; \vec{y}), & \text{else} \end{cases}$$

We abbreviate

$$h_{i,j,\ell}(\vec{x}, w, \mathsf{init}(c, F'(\vec{x}, w; \vec{y}))_1; \vec{y}, \vec{f_1}(\vec{x}, w; \vec{y}))$$

as $q_{i,j,\ell}(\vec{x}, w; \vec{y})$.

$$f_{j,\ell}(\vec{x}, \epsilon; \vec{y}) := g_{j,\ell}(\vec{x}, \epsilon; \vec{y})$$

$$f_{j,\ell}(\vec{x}, \mathsf{s}_i w; \vec{y}) := \begin{cases} q_{i,j,\ell}(\vec{x}, w; \vec{y}), & \text{if } F'(\vec{x}, w; \vec{y})_0 = 1 \\ f_{j,\ell}(\vec{x}, w; \vec{y}), & \text{else} \end{cases}$$

Finally, let us define the realisation function F with normal output. Let p be a polynomial such that $\langle 2, |w| \rangle \leq |p(w)|$. We define

$$r(\vec{x}, w; \vec{y}) := exp(w, \mathsf{init}(p(w); F'(\vec{x}, w; \vec{y}))_1;).$$

We let $F(\vec{x}, \mathsf{s}_i w; \vec{y})$ be given by the following case distinction.

$$\begin{cases} \mathsf{init}(c, F'(\vec{x}, \mathsf{s}_i w; \vec{y})), & \text{if } F'(\vec{x}, \mathsf{s}_i w; \vec{y})_0 = 1 \\ G(\vec{x}; \vec{y}), & \text{if } F'(\vec{x}, \mathsf{s}_i w; \vec{y})_0 = 2 \\ H_i(\vec{x}, r(\vec{x}, \mathsf{s}_i w; \vec{y}), \mathsf{init}(c, F'(\vec{x}, r(\vec{x}, \mathsf{s}_i w; \vec{y}); \vec{y})); \vec{y}), & \text{if } F'(\vec{x}, \mathsf{s}_i w; \vec{y})_0 = 3 \end{cases}$$

The correctness of the realisation functions is proved by an easy induction on the value of t in the standard model.

The other rules can be realised easily. □

From 19 and the previous theorem we derive the following lemma.

Lemma 24 *The theories* LogST *and* LogST′ *prove totality exactly for the logarithmic space computable functions.*

4.2. Formalising Clote's algebra for logspace

In the last section, we have defined a theory of strength logspace that formalises the algebra $\mathfrak{LS}$. Another possibility to produce a theory of this strength is to formalise the already mentioned algebra

$$[0, I, \mathsf{s}_0, \mathsf{s}_1, \mathsf{len}, \mathsf{bit}, \mathsf{e}, \times, COMP, CRN, SBRN],$$

where $SBRN$ denotes sharply bounded recursion. We give an induction principle capturing both, CRN and $SBRN$.

Definition 25 *For any positive formula A, we denote the formula A with each subformula of the form $t \in \mathsf{W}$ replaced by $t \leq_{\mathsf{W}} u$ by A^u.*

This notation has been introduced by Probst in [20].

Definition 26 *The theory* LogSB *is the theory* LogT *with the following modifications.*

- *The induction axiom is replaced by sharply bounded induction (SB-Ind) defined as follows, for A a positive formula and u a fresh variable* [8].

$$u \in \mathsf{W} \to \Big(A^{|u|}[\epsilon] \wedge (\forall x \in \mathsf{W})(A^{|u|}[x] \to A^{|u|}[\mathsf{s}_0 x] \wedge A^{|u|}[\mathsf{s}_1 x]) \to (\forall x \in \mathsf{W})(A^{|u|}[x])\Big)$$

- *We drop the axioms for bit, concatenation, multiplication, and eraser.*

Lemma 27 *The theory* LogSB *proves totality exactly for the logspace computable functions.*

Proof. The theory LogST' can simulate induction over formulas of the form $y \leq_{\mathsf{W}} |x|$, since it proves

$$x \in \mathsf{W} \to (y \leq_{\mathsf{W}} |x| \leftrightarrow y \in \mathsf{V} \wedge \mathsf{init}(x, y) = y)$$

because of the elementary properties and the axiom for init. This immediately implies the upper bound using theorem 16.

For the lower bound, we show that the logspace functions are provably total by induction on their complexity using the earlier mentioned function algebra

$$[\epsilon, I, \mathsf{s}_0, \mathsf{s}_1, \mathsf{len}, \mathsf{bit}, \times, \mathsf{e}, COMP, CRN, SBRN].$$

[8] Similar bounded induction schemes have been used by Kahle and Oitavem e.g. in [16], and Strahm and the author e.g. in [10, 11, 22].

The totality of $\epsilon, I, \mathsf{s}_0, \mathsf{s}_1, \mathsf{len}$ are clear. The totality of word multiplication follows as for LogST. For the definition of the eraser, we use the following auxiliary function h.

$$h(w) := \begin{cases} 1, & \text{if } w \text{ contains a 1} \\ \epsilon, & \text{else} \end{cases}$$

The totality of h is proved by $(SB - Ind)$. Then the eraser function is given by a term e fulfilling the following recursion equations.

$$\begin{aligned} \mathsf{e}(\epsilon;\,) &:= \epsilon \\ \mathsf{e}(\mathsf{s}_0 w;\,) &:= \begin{cases} \mathsf{s}_0\mathsf{e}(w), & \text{if } h(\mathsf{s}_0 w) = 1 \\ \mathsf{e}(w), & \text{else} \end{cases} \\ \mathsf{e}(\mathsf{s}_1 w;\,) &:= \mathsf{s}_1\mathsf{e}(w) \end{aligned}$$

Its totality is proved by V-induction.

Next, we show how the totality of the bit function is proved. We prove consecutively the totality of the functions $\dot{-}$, h, exp, bit^* defined in the lower bound proof of $\mathfrak{LG}$ on page 20. The totality of all of these functions is proved by V-induction in LogSB because only case distinction over elements of W are necessary. Then, we define bit using bit^*, and exp.

We sketch in the following how to deal with the recursion schemes. Assume that t_F represents a function defined by concatenation recursion. The totality of t_F is proved using (SB-Ind) with induction variable x for the formula $t_F x\vec{z} \in \mathsf{V}$ and the V-elimination rule, where we assume $\vec{z} \in \mathsf{W}$. To prove totality of a function F represented by t_F defined by sharply bounded recursion with bound B represented by t_B, we use induction over initial segments $\{w \in \mathbb{W} | w \subseteq a\}$ of W with induction variable x for the formula $t_F x\vec{z} \leq_{\mathsf{W}} |t_B a\vec{z}|$ [9]. This concludes the proof of the lower bound. □

5. Summary

We have shown that the introduction of two distinct word predicates W and V, first presented by Cantini [6], can be used nicely to reflect the different roles inputs play in weak recursion schemes. We propose a new reading of $t \in \mathsf{V}$ as "t is inaccessible" which is reflected by restricting the application of initial functions to elements of V.

[9] We can assume that the bound B is monotone

We presented a restricted case distinction for safe inputs and have shown that it allows a simple two-sorted characterisation of logspace. Interestingly, the restriction of case distinction has a similar effect on the safe recursion scheme as affinity restrictions (see Neergaard's [18]).

6. Acknowledgements

The author would like to thank the anonymous referees for helpful comments and suggestions that led to significant improvements of this paper. The author would also like to thank Prof. Thomas Strahm for his support.

References

[1] Beeson, M. J. *Foundations of Constructive Mathematics: Metamathematical Studies*. Springer, Berlin, 1985.

[2] Beeson, M. J. Proving programs and programming proofs. In *Logic, Methodology and Philosophy of Science VII*, Barcan Marcus et. al., Ed. North Holland, Amsterdam, 1986, pp. 51–82.

[3] Bellantoni, S. *Predicative Recursion and Computational Complexity*. PhD thesis, University of Toronto, 1992.

[4] Bellantoni, S., and Cook, S. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity 2* (1992), 97–110.

[5] Buss, S. R. *Bounded Arithmetic*. Bibliopolis, Napoli, 1986.

[6] Cantini, A. Polytime, combinatory logic and positive safe induction. *Archive for Mathematical Logic 41*, 2 (2002), 169–189.

[7] Clote, P. Computation models and function algebras. In *Handbook of Computability Theory*, E. Griffor, Ed. Elsevier, 1999, pp. 589–681.

[8] Clote, P., and Remmel, J., Eds. *Feasible Mathematics II*, vol. 13 of *Progress in Computer Science and Applied Logic*. Birkhäuser, Basel, 1995.

[9] Cook, S. A., and Nguyen, P. *Logical Foundations of Proof Complexity*. ASL Prespectives in Logic. Cambridge University Press, 2010.

[10] Eberhard, S. A feasible theory of truth over combinatory logic. Submitted, Oct. 2012.

[11] Eberhard, S., and Strahm, T. Unfolding feasible arithmetic and weak truth. In *Axiomatic Theories of Truth* (2012), T. Achourioti, H. Galinon, K. Fujimoto, and J. Martínez-Fernández, Eds., Logic, Epistemology and the Unity of Science, Springer. Being published.

[12] Feferman, S. A language and axioms for explicit mathematics. In *Algebra and Logic*, J. Crossley, Ed., vol. 450 of *Lecture Notes in Mathematics*. Springer, Berlin, 1975, pp. 87–139.

[13] Girard, J.-Y. *Proof Theory and Logical Complexitiy*. Bibliopolis, Napoli, 1987.

[14] Ishihara, H. Function algebraic characterizations of the polytime functions. *Computational Complexity 8* (1999), 346–356.

[15] Kahle, R., and Oitavem, I. An applicative theory for FPH. In *Proceedings Third International Workshop on Classical Logic and Computation CL&C* (2010), S. van Bakel, S. Berardi, and U. Berger, Eds., vol. 47 of *EPTCS*.

[16] Kahle, R., and Oitavem, I. Applicative theories for the polynomial hierarchy of time and its levels. Accepted for publication in Annals of Pure and Applied Logic, 2012.

[17] Krajíček, J. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*, vol. 60 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1995.

[18] Møller Neergaard, P. A functional language for logarithmic space. In *Asian Symposium on Programming Languages and Systems*. 2004, pp. 311–326.

[19] Oitavem, I. Logspace without Bounds. In *Ways of proof theory*, R. Schindler, Ed. Ontos Verlag, 2010, pp. 349–356.

[20] Probst, D. The provably terminating operations of the subsystem PETJ of explicit mathematics. *Annals of Pure and Applied Logic 162*, 11 (2011), 934–947.

[21] Strahm, T. *Proof-theoretic Contributions to Explicit Mathematics*. Habilitationsschrift, University of Bern, 2001.

[22] Strahm, T. Theories with self-application and computational complexity. *Information and Computation 185* (2003), 263–297.

[23] Strahm, T. Weak theories of operations and types. In *Ways of Proof Theory*, R. Schindler, Ed. Ontos Verlag, 2010, pp. 441–468.